AFRL-IF-RS-TR-2006-89
Final Technical Report
March 2006

SEMANTIC WEB TECHNOLOGIES FOR MOBILE CONTEXT-AWARE SERVICES

Carnegie Mellon University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J391

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2006-89 has been reviewed and is approved for publication.

APPROVED: /s/

MARK GORNIAK
Project Engineer

FOR THE DIRECTOR: /s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 074-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** MARCH 2006	**3. REPORT TYPE AND DATES COVERED** Final Jan 2003 – Jan 2005	
4. TITLE AND SUBTITLE SEMANTIC WEB TECHNOLOGIES FOR MOBILE CONTEXT-AWARE SERVICES			**5. FUNDING NUMBERS** C - F30602-03-0025 PE - 62301E PR - DAML TA - 00 WU - 21
6. AUTHOR(S) Norman M. Sadeh			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ISRI – School of Computer Science Carnegie Mellon University 5000 Forbes Avenue Pittsburgh Pennsylvania 15213			**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714; AFRL/IFSB 525 Brooks Road Rome New York 13441-4505			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2006-89
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Mark Gorniak/IFSB/(315) 330-7724 Mark.Gorniak@rl.af.mil			
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 Words)*

The emergence of Semantic Web Services and automated service discovery, access and composition functionality will enable higher levels of interoperability and automation across a broad range of contexts (e.g. intra- and inter-organization collaboration, dynamic virtual enterprises, coalition forces, pervasive computing, etc.). The objective of this project was to demonstrate, validate and disseminate Semantic Web technologies aimed at reconciling the interoperability and confidentiality requirements associated with these scenarios, focusing in particular on mobile, context-aware services. This final performance report provides a comprehensive, cumulative and substantive summary of the progress and significant accomplishments achieved during the course of this project.

14. SUBJECT TERMS Semantic Web, Context Awareness, Web Services, Privacy, Mobility, Rules			**15. NUMBER OF PAGES** 95
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Table of Contents

List of Appendixes

List of Figures

1 Summary

With hundreds of millions of Internet-enabled mobile devices, the mobile Internet is opening the door to a slew of new mobile applications and services that will assist users as they engage in time-critical, goal-driven tasks. Yet today, these devices and associated mobile scenarios give rise to particularly challenging usability issues. Overcoming these challenges will likely require the development of applications that can dynamically recognize and adapt to the context in which their users operate (e.g. location, activities and tasks, surrounding environment, organizational or social context, etc.). The objective of this project was to demonstrate and evaluate the use of semantic web technologies (1) in **facilitating the development and maintenance of context-aware applications** and (2) **in capturing and enforcing associated security and privacy/confidentiality policies**. This included demonstrating the **relevance** of these technologies in the context of **both civilian and DoD-oriented scenarios**. In the process, this project also provided a vehicle to test drive early semantic web languages and contributed to recommendations for extending these languages in several areas (semantic web services, rules and privacy and security policies).

The primary findings of this project are:

- Semantic web technologies can play a key role **in facilitating the rapid development and maintenance of context-aware applications**, as demonstrated by the project's experience in several application domains (e.g. MyCampus domain aimed at enhancing everyday campus life through an open collection of context-aware applications, SONAT context-aware message filtering and delivery application, context-aware museum tour guide applications)
- When extended to support **rules,** semantic web technologies provide a particularly powerful framework to capture and enforce **rich security and privacy policies** (including **context-sensitive policies**)

Key additional accomplishments include:

- **Extensions of the OWL language** to represent rules and development of associated reasoning functionality
- **Public release of two software tools** (ROWL engine and Semantic eWallet) on SemWebCentral (http://www.semwebcentral.org/) – about 1,000 page views and 50 downloads in the first three months
- Demonstrations on **both civilian and DoD-oriented scenarios** (e.g. MyCampus, SONAT, CoSAR-TS)
- Contributions to **semantic web service architecture** working group and **semantic web rules working groups**
- Development of a **semantic web architecture for context-aware service provisioning and privacy** that is influencing a number of ongoing R&D efforts in the telecommunications industry (e.g. work at Motorola, Nokia, NTTDoCoMo, Siemens, etc.)
- Numerous **publications, presentations, and other education and dissemination** activities

2 Introduction

The emergence of Semantic Web Services and automated service discovery, access and composition functionality will enable higher levels of interoperability and automation across a broad range of contexts (e.g. **intra- and inter-organization collaboration, dynamic virtual enterprises, coalition forces, pervasive computing,** etc.). The broad objective of this project was to **demonstrate, validate and disseminate Semantic Web technologies aimed at reconciling the interoperability and confidentiality** requirements associated with these scenarios, focusing in particular on **mobile, context-aware services**.

Specifically, application developers are looking for ways to provide users with added levels of convenience and ease of use through functionality that is capable of capturing the context within which they operate. This may involve knowing where the user is located, the task she is currently engaged in, her eating preferences, who her colleagues are as well as a variety of other contextual attributes. While there are many sources of contextual information, they tend to vary from one user to another and also over time. Different users may rely on different location tracking functionality provided by different cell phone operators; they may use different calendar systems, *etc.* Traditionally, context-aware applications and services have been hardwired to predefined sources of contextual information (e.g. relying on a particular set of sensors and protocols to track a user's locations). As a result, they remain prohibitively expensive to build and maintain and are few and between. Instead, in our project we have shown that, using Semantic Web technologies, it is possible to develop significantly more open environments, where context-aware applications can automatically discover and access a user's personal resources such as her calendar or location tracking functionality. This is done by viewing sources of contextual information (or personal resource) as Semantic Web services that can automatically be identified (and hence "re-used") thanks to rich semantic profiles. As users and organizations start exposing functionality in the form of semantic web services, it is critical that they also be able to control who has access to this information and under what conditions. A user may be willing to let her colleagues see where she is or access her calendar activities between 8am and 5pm on weekdays but not over the weekend. Similarly, when it comes to **sharing sensitive contextual information with allies and other coalition forces, the DoD needs to be able to enforce rich, context-sensitive policies that take into account factors such as the nature of each particular alliance or the relevance of requested information to specific joint missions**. Capturing the requirements associated with these scenarios has led our project to advance the state of the art in Semantic Web technologies in several areas:

- Extending emerging Semantic Web languages and reasoning technologies to **capture rules** (e.g. security and privacy policies, user preferences, service identification rules, etc.)
- Developing a Semantic Web architecture for **context-aware service provisioning**
- Developing Semantic Web technologies and tools to enforce **rich, context-sensitive security and privacy policies**

The remainder of this report is organized as follows. The following section summarizes the methods, assumptions and procedures used in our work. Section 4 summarizes and discusses important project results. Conclusions are presented in Section 5. References are provided in Section 6. Additional technical details are discussed in an Appendix at the end of this report.

3 Methods, Assumptions and Procedures

3.1 A Semantic Web Architecture for Context-Aware Service Provisioning

We have developed a Semantic Web architecture for context-aware service provisioning, where sources of contextual information are modeled as semantic web services. Entities such as people, groups or organizations control access to their contextual information via Semantic e-Wallets that serve a dual role of clearinghouse and gatekeeper to this information. As users subscribe to or simply access new context-aware applications, these applications attempt to obtain relevant information by querying their users' e-Wallets as well as possibly those of others (e.g. meeting scheduling requires accessing calendar information for multiple users). During the course of the project, this architecture was validated in several domains:

- *MyCampus*: A Semantic Web environment aimed at enhancing everyday campus life at Carnegie Mellon University through deployment of a growing collection of context-aware applications that users can access from PDAs and other access devices over the campus's 802.11 wireless LAN (see Figure 1, 2, 3 and 4). Over a dozen context-aware applications have been developed (context-aware restaurant concierge, meeting scheduler with obfuscation rules, people locator with privacy preferences, location sensitive services such as movie guide, weather service, etc., context-aware message filtering and delivery services, remote slide viewing with access control rules, P2P context-aware poster application, context-aware reminder applications, etc.). Detailed evaluation has been conducted with real-users on campus over periods of several days [SCV+03,GS03,Sad03,GS04a,SG05]. Extensions of the environment to support service composition functionality are also discussed in [SSG04].

- *The Semantic Operational Net Assessment Tool (SONAT)* developed by BBN to showcase different DAML tools and technologies in a DoD context (Figure 5)
- *The Coalition Search and Rescue – Task Support (CoSAR-TS)* scenario developed through a collaboration between AIAI (Edinburgh), the University of West Florida, BBN and SPAWAR.
- *A prototype context-aware Museum tour guide* developed for the National Museum of Natural Science in Taiwan – one of the island's largest museums with over 3 million visitors per year (Figure 6) [CHGS05]

- *An office environment* relying on the Aura context-aware infrastructure (Figure 7) [MJH+04]

While it is difficult to objectively evaluate reductions in development and maintenance costs afforded by our Semantic Web framework, especially in the context of early prototyping efforts, anecdotal evidence collected through the successive development and refinement of a series of context-aware applications in MyCampus suggests that the gains are rather significant. Applications that would typically have required a week or more of development could typically be developed in a matter of a day or two – sometimes less.

These benefits are mainly attributed to the wrapping of sources of contextual information as web services that can readily be shared across a growing collection of context-sensitive applications and to the ability of our architecture to quickly capture complex, context-sensitive privacy/confidentiality policies that can refer to any concept in shared contextual ontologies.

Our Semantic Web architecture for context-aware service provisioning has influenced ongoing research efforts at a number of companies in the telecom sector (e.g. Nokia, Ericsson, Motorola, Siemens, NTTDoCoMo, Lucent, etc.), as acknowledged in their publications [Mobi].

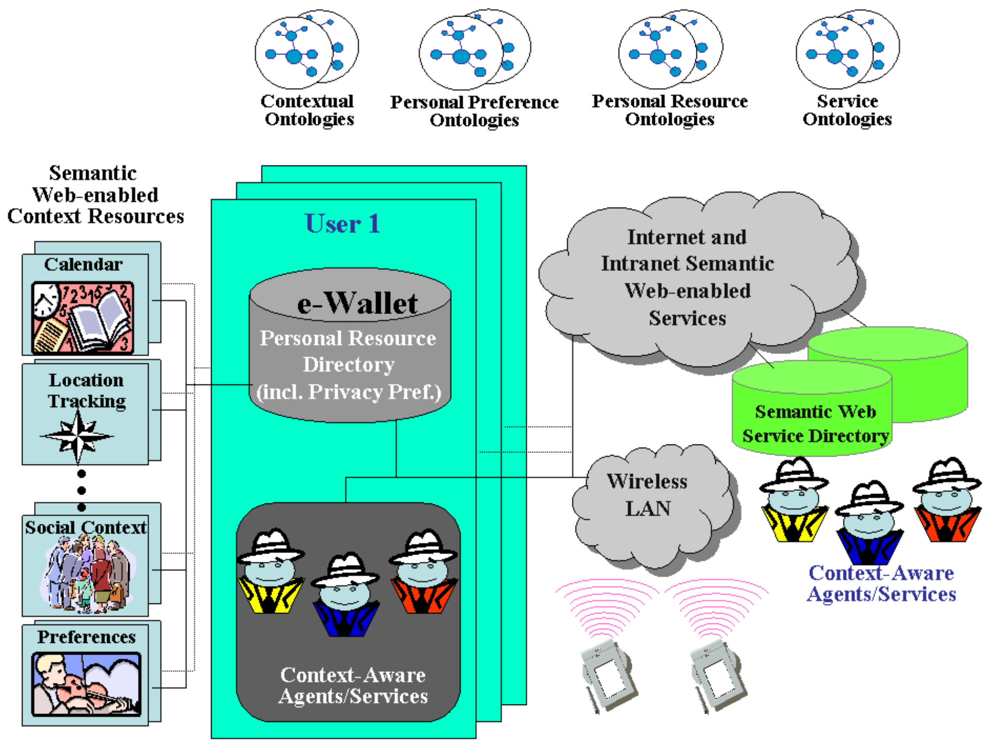

Contextual Ontologies
Personal Preference Ontologies
Personal Resource Ontologies
Service Ontologies
Semantic Web-enabled Context Resources
Calendar
Location Tracking
Social Context
Preferences
User 1
e-Wallet
Personal Resource Directory (incl. Privacy Pref.)
Context-Aware Agents/Services
Internet and Intranet Semantic Web-enabled Services
Semantic Web Service Directory
Wireless LAN
Context-Aware Agents/Services

Figure 1. MyCampus instantiation of our Semantic Web Architecture for Context-Aware Service Provisioning

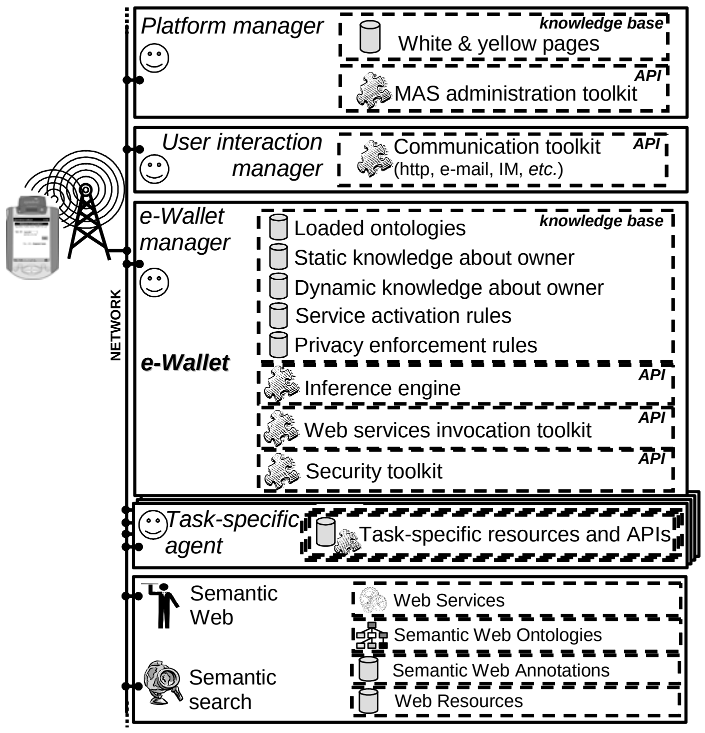
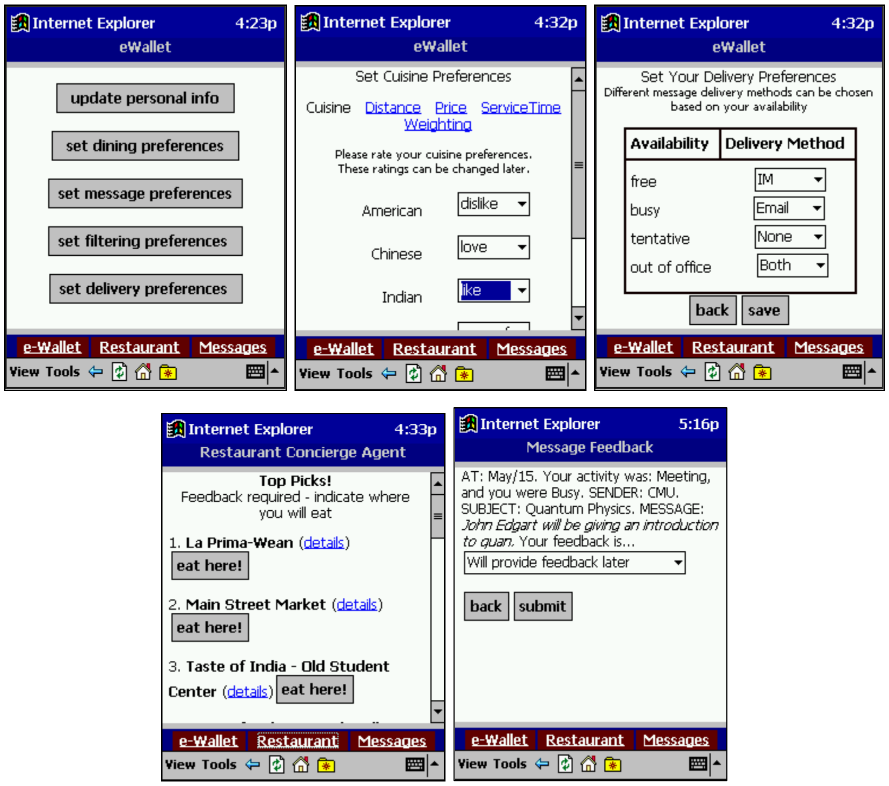


Figure 2. *MyCampus* architecture: user perspective - smiley faces are agents.


Internet Explorer 4:23p
eWallet

update personal info
set dining preferences
set message preferences
set filtering preferences
set delivery preferences

e-Wallet Restaurant Messages
View Tools

Internet Explorer 4:32p
eWallet

Set Cuisine Preferences
Cuisine Distance Price ServiceTime Weighting

Please rate your cuisine preferences.
These ratings can be changed later.

American	dislike
Chinese	love
Indian	like

e-Wallet Restaurant Messages
View Tools

Internet Explorer 4:32p
eWallet

Set Your Delivery Preferences
Different message delivery methods can be chosen based on your availability

Availability	Delivery Method
free	IM
busy	Email
tentative	None
out of office	Both

back save

e-Wallet Restaurant Messages
View Tools

Internet Explorer 4:33p
Restaurant Concierge Agent

Top Picks!
Feedback required - indicate where you will eat

1. La Prima-Wean (details)
eat here!

2. Main Street Market (details)
eat here!

3. Taste of India - Old Student Center (details) eat here!

e-Wallet Restaurant Messages
View Tools

Internet Explorer 5:16p
Message Feedback

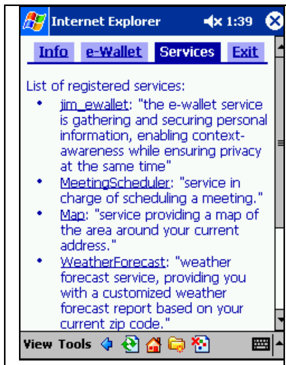
AT: May/15. Your activity was: Meeting, and you were Busy. SENDER: CMU. SUBJECT: Quantum Physics. MESSAGE: *John Edgart will be giving an introduction to quan.* Your feedback is...

Will provide feedback later

back submit

e-Wallet Restaurant Messages
View Tools

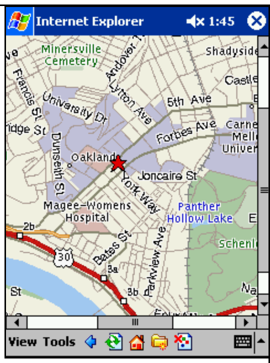
Figure 3. MyCampus screenshots, including the e-Wallet (top), a context-aware Restaurant Concierge Agent (bottom left) and a context-aware Message Filtering Agent (bottom right)



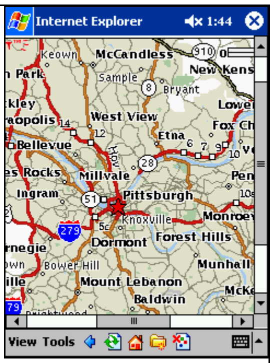
List of registered services:

- jim_ewallet: "the e-wallet service is gathering and securing personal information, enabling context-awareness while ensuring privacy at the same time"
- MeetingScheduler: "service in charge of scheduling a meeting."
- Map: "service providing a map of the area around your current address."
- WeatherForecast: "weather forecast service, providing you with a customized weather forecast report based on your current zip code."

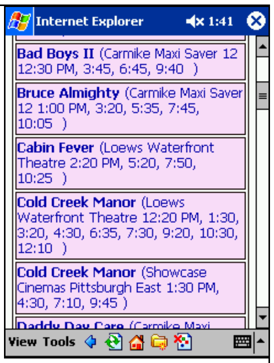
List of Services



People Finder (1)

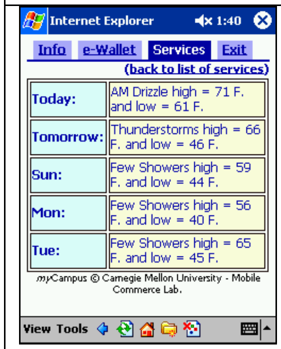


People Finder (2)



Bad Boys II (Carmike Maxi Saver 12 12:30 PM, 3:45, 6:45, 9:40)
Bruce Almighty (Carmike Maxi Saver 12 1:00 PM, 3:20, 5:35, 7:45, 10:05)
Cabin Fever (Loews Waterfront Theatre 2:20 PM, 5:20, 7:50, 10:25)
Cold Creek Manor (Loews Waterfront Theatre 12:20 PM, 1:30, 3:20, 4:30, 6:35, 7:30, 9:20, 10:30, 12:10)
Cold Creek Manor (Showcase Cinemas Pittsburgh East 1:30 PM, 4:30, 7:10, 9:45)
Daddy Day Care (Carmike Maxi

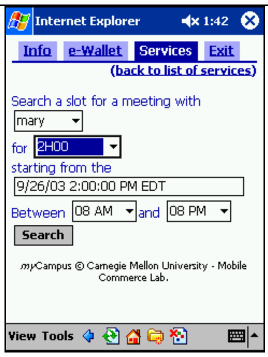
Loc.-Aw. Movie Guide



Today:	AM Drizzle high = 71 F. and low = 61 F.
Tomorrow:	Thunderstorms high = 66 F. and low = 46 F.
Sun:	Few Showers high = 59 F. and low = 44 F.
Mon:	Few Showers high = 56 F. and low = 40 F.
Tue:	Few Showers high = 65 F. and low = 45 F.

*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

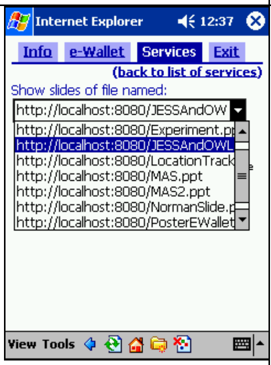
Loc.-Aware Weather



Search a slot for a meeting with mary
for 2H00
starting from the 9/26/03 2:00:00 PM EDT
Between 08 AM and 08 PM

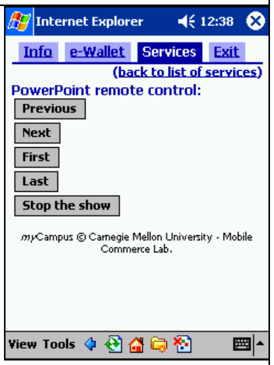
*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

Meeting Scheduler



Show slides of file named:
http://localhost:8080/JESSAndOW
http://localhost:8080/Experiment.p
http://localhost:8080/JESSAndOWL
http://localhost:8080/LocationTrack
http://localhost:8080/MAS.ppt
http://localhost:8080/MAS2.ppt
http://localhost:8080/NormanSlide.p
http://localhost:8080/PosterEWallet

Slide Browsing (1)

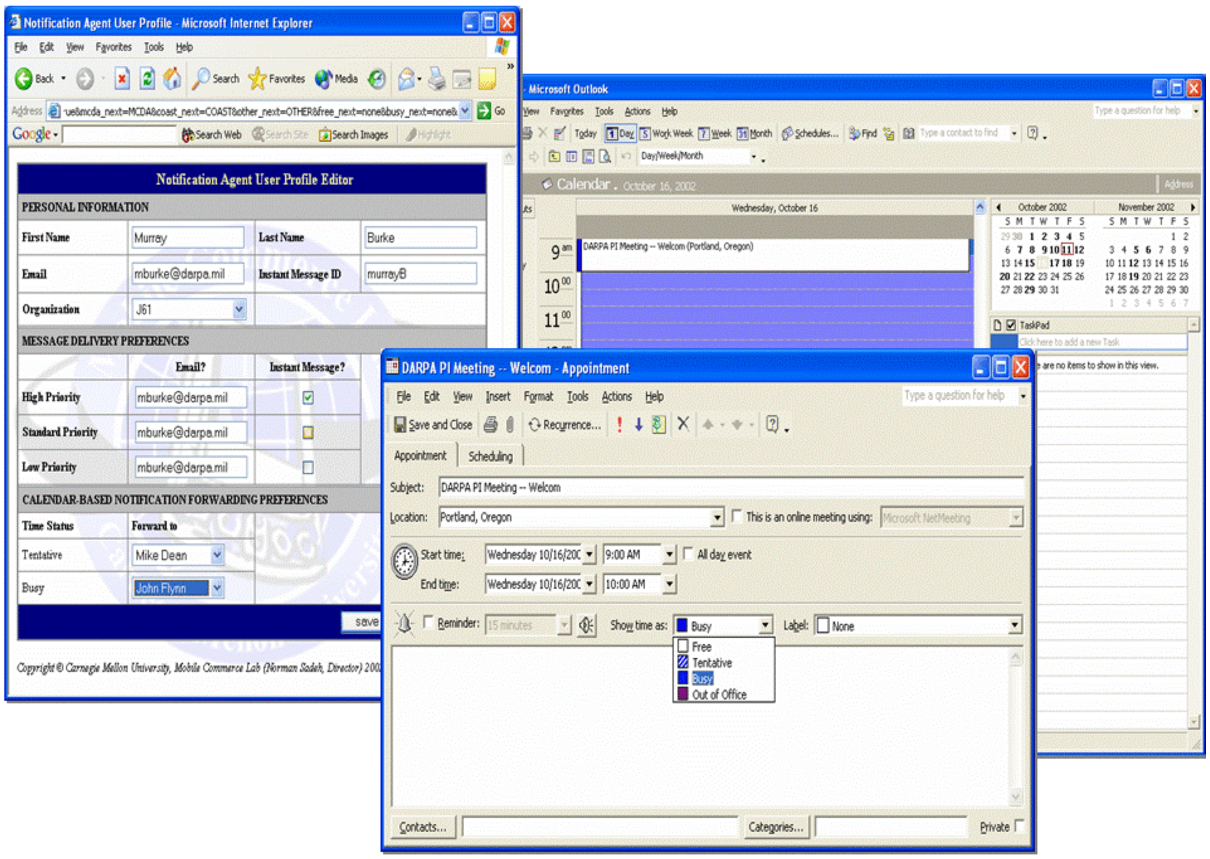


PowerPoint remote control:

*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

Slide Browsing (2)

Figure 4. MyCampus: Additional screen shots



Notification Agent User Profile Editor

PERSONAL INFORMATION

First Name	Murray	Last Name	Burke
Email	mburke@darpa.mil	Instant Message ID	murrayB
Organization	J61		

MESSAGE DELIVERY PREFERENCES

	Email?	Instant Message?
High Priority	mburke@darpa.mil	✓
Standard Priority	mburke@darpa.mil	☐
Low Priority	mburke@darpa.mil	☐

CALENDAR-BASED NOTIFICATION FORWARDING PREFERENCES

Time Status	Forward to
Tentative	Mike Dean
Busy	John Flynn

Copyright © Carnegie Mellon University, Mobile Commerce Lab (Norman Sadeh, Director) 200

DARPA PI Meeting -- Welcom (Portland, Oregon)

DARPA PI Meeting -- Welcom - Appointment

Subject: DARPA PI Meeting -- Welcom

Location: Portland, Oregon

This is an online meeting using: Microsoft NetMeeting

Start time: Wednesday 10/16/200 9:00 AM

End time: Wednesday 10/16/200 10:00 AM

All day event ☐

Reminder: ☐ 15 minutes

Show time as: Busy

Label: None

Figure 5. SONAT: CMU's Context-Aware Message Delivery Agent

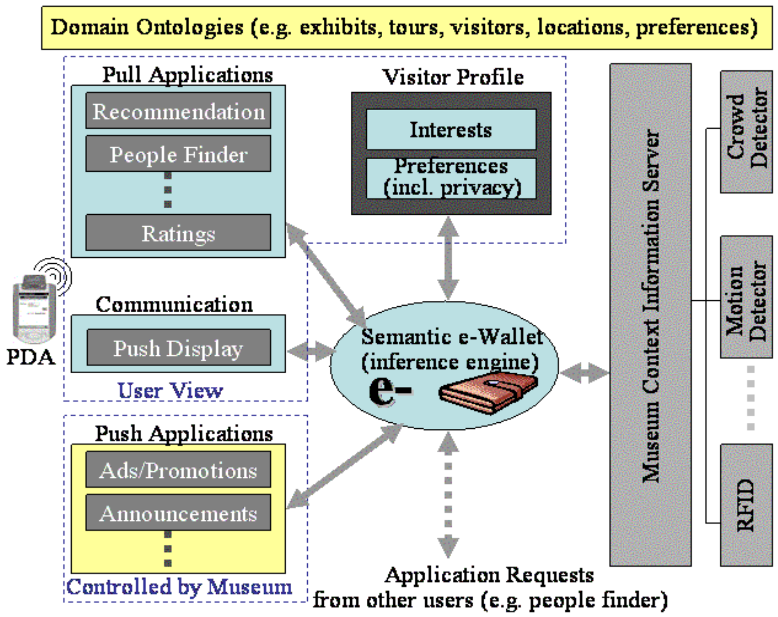
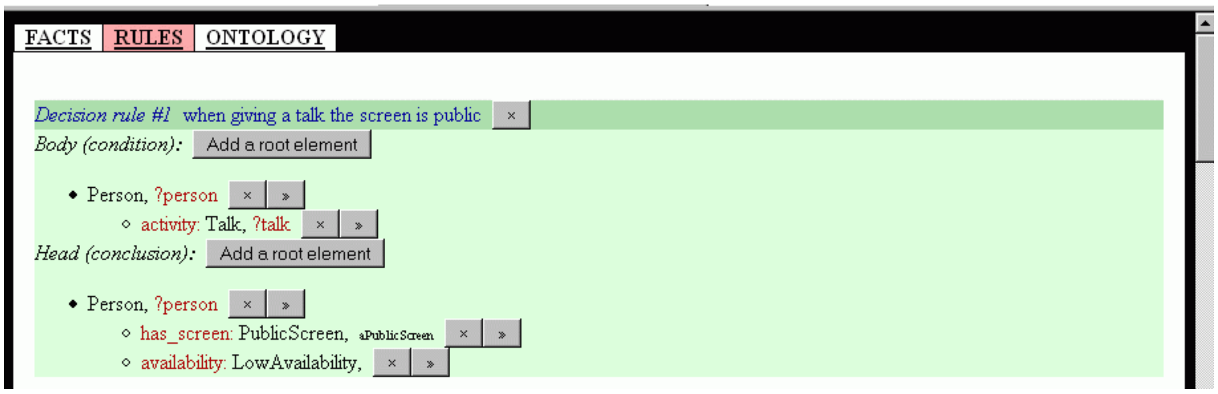

Domain Ontologies (e.g. exhibits, tours, visitors, locations, preferences)
Pull Applications
Recommendation
People Finder
Ratings
Visitor Profile
Interests
Preferences (incl. privacy)
PDA
Communication
Push Display
User View
Semantic e-Wallet (inference engine)
e-
Push Applications
Ads/Promotions
Announcements
Controlled by Museum
Application Requests from other users (e.g. people finder)
Museum Context Information Server
Crowd Detector
Motion Detector
RFID

Figure 6. Architecture of Context-Aware Museum Tour Guide


FACTS RULES ONTOLOGY

Decision rule #1 when giving a talk the screen is public ×

Body (condition): Add a root element

- Person, ?person × »
  - activity: Talk, ?talk × »

Head (conclusion): Add a root element

- Person, ?person × »
  - has_screen: PublicScreen, aPublicScreen × »
  - availability: LowAvailability, × »

Figure 7. Rule Editor for Semantic Web Context-Aware Messaging Application integrated into the Aura smart office environment

3.2 ROWL – Rule Extension of OWL

Another important part of our work has been the development of extensions of OWL to capture and exchange rule descriptions. This work resulted in ROWL ("Rule Extension of OWL") [GS04a,GS04b]. Rules arise in a variety of contexts (e.g. B2B contracts, negotiations, security, privacy, workflow management, context-aware computing, etc.). Rules in ROWL are Horn clauses of the type: **Body** => **Head.** They can refer to ontologies and annotations and contain variables. Rules are defined as an OWL ontology. A Rule construct includes a Label (used to assign a name to a rule), a Head and a Body (e.g. Figure 8).

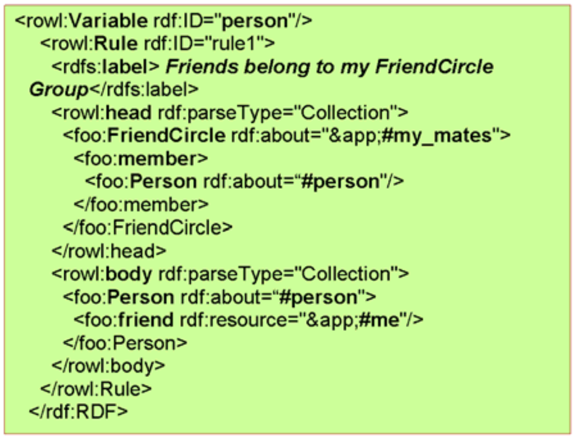

```
<rowl:Variable rdf:ID="person"/>
  <rowl:Rule rdf:ID="rule1">
    <rdfs:label> Friends belong to my FriendCircle
 Group</rdfs:label>
    <rowl:head rdf:parseType="Collection">
      <foo:FriendCircle rdf:about="&app;#my_mates">
        <foo:member>
          <foo:Person rdf:about="#person"/>
        </foo:member>
      </foo:FriendCircle>
    </rowl:head>
    <rowl:body rdf:parseType="Collection">
      <foo:Person rdf:about="#person">
        <foo:friend rdf:resource="&app;#me"/>
      </foo:Person>
    </rowl:body>
  </rowl:Rule>
 </rdf:RDF>
```

Figure 8. Sample rule in ROWL

ROWL was first released in early 2004 through Carnegie Mellon University's MyCampus website and later through SemWebCentral. The release includes (Figure 9):

- RDF/OWL meta-model in CLIPS and more specifically into native JESS code
- XSLT stylesheets to transform OWL ontologies into CLIPS and more specifically into native JESS code

- XSLT stylesheets to transform RDF instances (annotations) into CLIPS and more specifically into native JESS code
- XSLT stylesheets to transform ROWL Rules into CLIPS, and more specifically into native JESS code
- Hooks to upload facts and rules into reasoning engines other than JESS

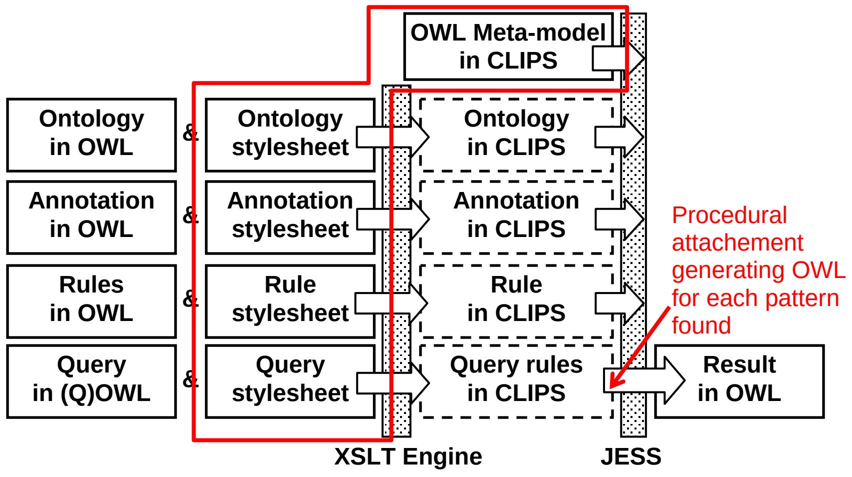
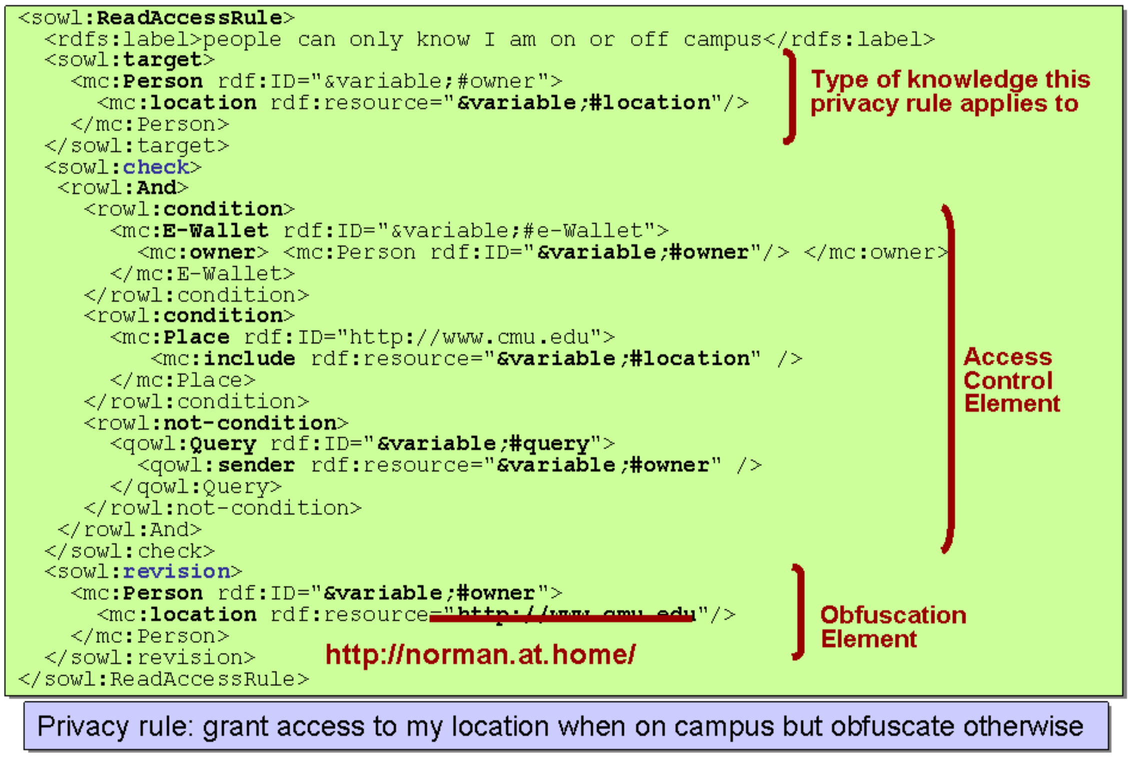


Figure 9. ROWL Semantic Web Inference Engine Architecture

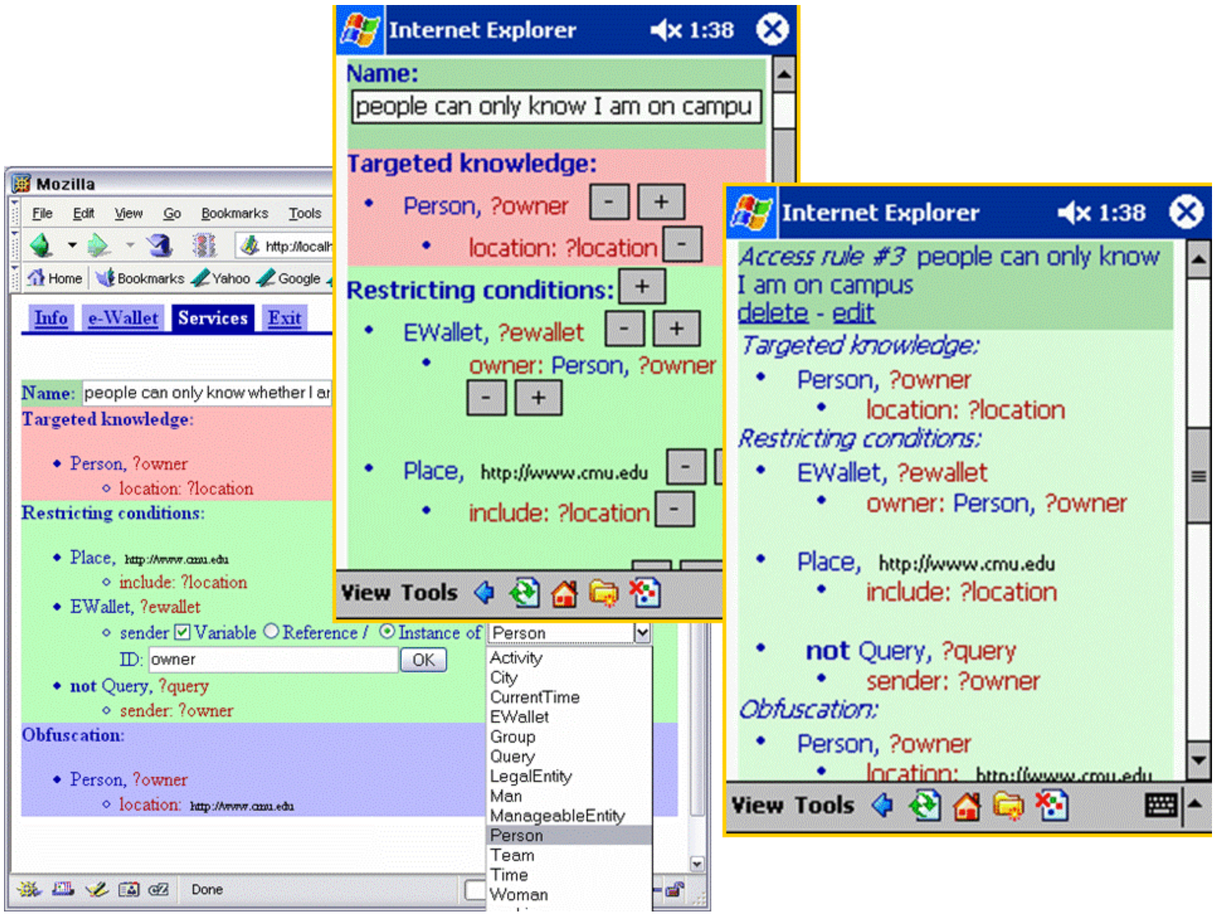


```
<sowl:ReadAccessRule>
  <rdfs:label>people can only know I am on or off campus</rdfs:label>
  <sowl:target>
    <mc:Person rdf:ID="&variable;#owner">
      <mc:location rdf:resource="&variable;#location"/>
    </mc:Person>
  </sowl:target>
  <sowl:check>
   <rowl:And>
     <rowl:condition>
       <mc:E-Wallet rdf:ID="&variable;#e-Wallet">
         <mc:owner> <mc:Person rdf:ID="&variable;#owner"/> </mc:owner>
       </mc:E-Wallet>
     </rowl:condition>
     <rowl:condition>
       <mc:Place rdf:ID="http://www.cmu.edu">
          <mc:include rdf:resource="&variable;#location" />
       </mc:Place>
     </rowl:condition>
     <rowl:not-condition>
       <qowl:Query rdf:ID="&variable;#query">
         <qowl:sender rdf:resource="&variable;#owner" />
       </qowl:Query>
     </rowl:not-condition>
   </rowl:And>
  </sowl:check>
  <sowl:revision>
    <mc:Person rdf:ID="&variable;#owner">
      <mc:location rdf:resource=~~"http://www.cmu.edu~~"/>
    </mc:Person>
  </sowl:revision>      http://norman.at.home/
</sowl:ReadAccessRule>
```

Privacy rule: grant access to my location when on campus but obfuscate otherwise

Figure 10. Sample context-sensitive privacy/confidentiality rule with both access control and obfuscation elements).

This engine was further extended to support the specification of a rich set of context-sensitive privacy and security policies. This includes context-sensitive access control policies as well as context-sensitive obfuscation policies that control the (in)accuracies at which information is released in response to queries. Figure 10 provides an example of context-sensitive privacy policy that includes both access control and obfuscation restrictions. The “target” part of the rule refers to the knowledge this privacy rule applies to (i.e. the location of the e-Wallet’s owner in this case). The “check” element contains access control restrictions (i.e. the owner has to be in a location that is on campus and the sender of the query is not the owner of the e-Wallet). The “revision” element contains obfuscation rules (e.g. only releasing information in terms of whether the owner is on or off campus or, as suggested with the “’cross-out”, falsifying the owner’s location and pretending that he is at home


Mozilla
File Edit View Go Bookmarks Tools
Home Bookmarks Yahoo Google
Info e-Wallet Services Exit
Done
Internet Explorer 1:38
View Tools

Name: people can only know whether I ar

Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- Place, http://www.cmu.edu
  - include: ?location
- EWallet, ?ewallet
  - sender ☑ Variable ○ Reference / ⦿ Instance of Person
    ID: owner OK
- not Query, ?query
  - sender: ?owner

Obfuscation:
- Person, ?owner
  - location: http://www.cmu.edu

Activity
City
CurrentTime
EWallet
Group
Query
LegalEntity
Man
ManageableEntity
Person
Team
Time
Woman

Name:
people can only know I am on campu

Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- EWallet, ?ewallet
  - owner: Person, ?owner
- Place, http://www.cmu.edu
  - include: ?location

Access rule #3 people can only know I am on campus
delete - edit

Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- EWallet, ?ewallet
  - owner: Person, ?owner
- Place, http://www.cmu.edu
  - include: ?location
- not Query, ?query
  - sender: ?owner

Obfuscation:
- Person, ?owner
  - location: http://www.cmu.edu

Figure 11. Editor for context-sensitive privacy and security policies

A rule editor was also developed to enable system administrators to rapidly specify/edit context-sensitive privacy and security policies, as they relate to any shared ontologies (Figure 11) and a special semantic web reasoning engine, referred to as a “Semantic e-Wallet” was developed to enforce these policies (see Section 3.3).

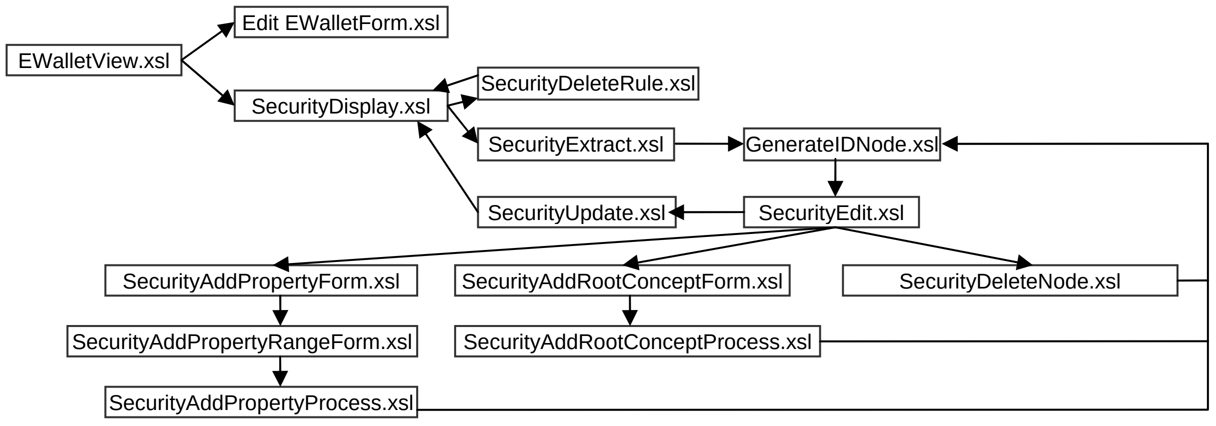


Figure 12. Editor for context-sensitive privacy and security policies: Flow diagram

The rule editor allows users to create new rules as well as edit and delete existing ones. The editor draws directly on available ontologies (ontologies loaded into the e-Wallet), enabling users to express any privacy/security rules they want as they relate to concepts and properties defined in these ontologies. It takes into account the OWL meta-model as the user edits rules. For instance, it will restrict the instantiation of a given concept to be within the range of a given property, as specified using the OWL "ObjectProperty" construct. As shown in Figure 12 editing operations are specified through external XSLT stylesheets

3.3 Semantic e-Wallets

Semantic e-Wallets are a central element of our Semantic Web architecture for context-aware service provisioning. They provide a unified front-end to the information and services of individual users as well as possibly groups of users and organizations. As such they act both as clearinghouses and gatekeepers to information and services, helping answer queries that require accessing this information and, in the process, enforcing relevant privacy and security policies. This includes context-sensitive privacy and security policies. To enforce these policies, our Semantic e-Wallet uses a multi-layer reasoning framework that differentiates between different types of triples/facts (Figure 13):

- Core Layer: At the most basic level, the e-Wallet's knowledge includes an OWL meta-model – required to interpret OWL statements. In addition, it maintains both static (context-independent) and dynamic (context-dependent) knowledge about the user *i.e.* rules completing the base with additional knowledge. This knowledge is obtained by loading available annotations about the user along with relevant ontologies and is currently completed using forward chaining reasoning – to avoid having to infer the same facts over and over again. Knowledge in this layer is represented using a (core) triple template:

$$(\text{predicate, subject, object})_{\text{TRIPLE}}$$

- The Service Layer completes the e-Wallet's core knowledge with invocation rules that map information retrieval goals about contextual attributes onto external service invocations. These are modeled as backward chaining rules. Given an information retrieval goal such as "Give me Fabien's location", they help identify and invoke one or more relevant information resources, each modeled as a Web service. Knowledge in this layer is represented using a special type of triple called "service triple" denoted:

  $$(\text{predicate, subject, object})_{\text{SERVICE TRIPLE}}$$

  Service triples reside in the service layer and are created in either of two ways. They can result from the migration of a triple from the core layer or from the activation of an invocation rule (e.g. an assertion about Fabien's location as returned by a call to a location tracking service). Migration between the core layer and the service layer is implemented by rules specifying that any (core) triple can be used to generate an equivalent service triple.

- The outer layer is referred to as the Privacy Layer, as this is where privacy enforcing rules are applied. Asssertions in this layer are represented as another special type of triple called "authorized triple": $(\text{predicate, subject, object})_{\text{AUTHORIZED TRIPLE}}$

  Only authorized triples can be sent in response to queries. Authorized triples are generated by applying privacy enforcing rules to service triples, thereby ensuring that information about the user is only disclosed to authorized parties and in accordance with relevant obfuscation rules. Privacy enforcing rules are encoded as backward chaining rules. These rules map needs for authorized triples onto needs for service triples to be post-processed subject to the privacy enforcing rules. Upon receiving an incoming query, the e-Wallet generates a need for one or more authorized triples. This need in turn typically triggers needs for service triples and core triples, eventually resulting in either (a) the generation of authorized triples that can be returned in response to the query or in (b) an exception, if the query is found to be unallowable (e.g. an unauthorized party requesting your location or trying to schedule a meeting in your calendar).

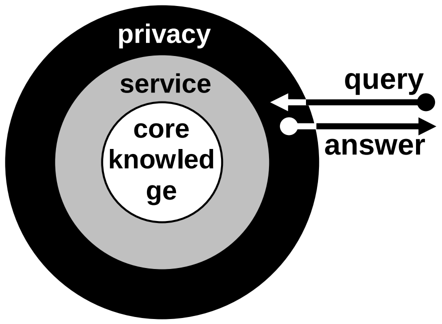


Figure 13. Three-layer initial architecture

Except for the core layer, the other layers are initially empty of facts. The arrival of a query/request will result in the generation of facts in these layers through backward chaining.

For a while, the e-Wallet's access control functionality was limited to supporting resource/service queries and the three layers were sufficient. More recently, we extended the architecture to also support the updating of static knowledge stored in the e-Wallet.

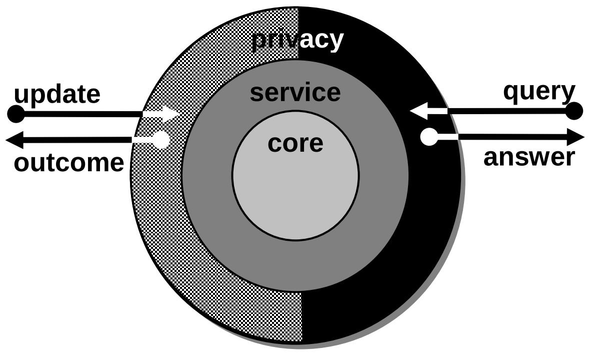


Figure 14. Four-layer current architecture

The privacy layer is now split into two (Figure 14): (predicate, subject, object)$_{\text{AUTHORIZED TRIPLE}}$ triples are used for query solving and (predicate, subject, object)$_{\text{UPDATE\_REQUEST\_TRIPLE}}$ triples are used to represent the arrival of requests for updates. Privacy rules for "Write Access" management handle these (p, s, o)$_{\text{UPDATE\_REQUEST\_TRIPLE}}$ triples and transform them into (predicate, subject, object)$_{\text{TRIPLE}}$ triples if allowed by the privacy preferences.

The current implementation of our e-Wallet is based on JESS [JESS], a high-performance Java-based rule engine that supports both forward and backward chaining – the latter by reifying "needs for facts" as facts themselves, which in turn trigger forward-chaining rules. In other words, the e-Wallet is implemented as an extension of the ROWL reasoning engine introduced in Figure 9. The e-Wallet's knowledge base is initialized with an OWL Meta-model in CLIPS that contains:

- A model of RDF [RDF] triples as a template for unordered facts,
- A model of specialized triples used in our layers (initial triples, core triples, service triples and authorized triples) along with associated migration rules between the layers, and
- The RDFS and OWL Lite meta-model assertions and rules.

We distinguish between four families of rule languages and a very simple query language (Figure 15):

- The ROWL (Rules in OWL) language is used to encode forward-chaining rules completing the base with additional knowledge that may be implicit in the current set of facts e.g. definitions (two persons belonging to the same group are colleagues), preferences (when in class I don't want to be disturbed), *etc.*
- The WOWL (Web services in OWL) language is used to encode backward-chaining rules describing the type of knowledge that can be obtained by calling a service, the knowledge needed to call that service and the way it should be called.

- The SOWL (Security in OWL) language is used to encode backward-chaining rules describing the privacy preferences of a user *i.e.* the conditions under which a piece of knowledge can be accessed or modified.
- The QOWL (Query in OWL) language is used to encode a backward-chaining rule describing a query. The body of the rule is given by the pattern in QOWL *i.e.* an OWL pattern with variables identified by a specific namespace. This language is also used to encode backward-chaining rules describing an update.

Details on the syntax of these languages are provided in Appendix A.

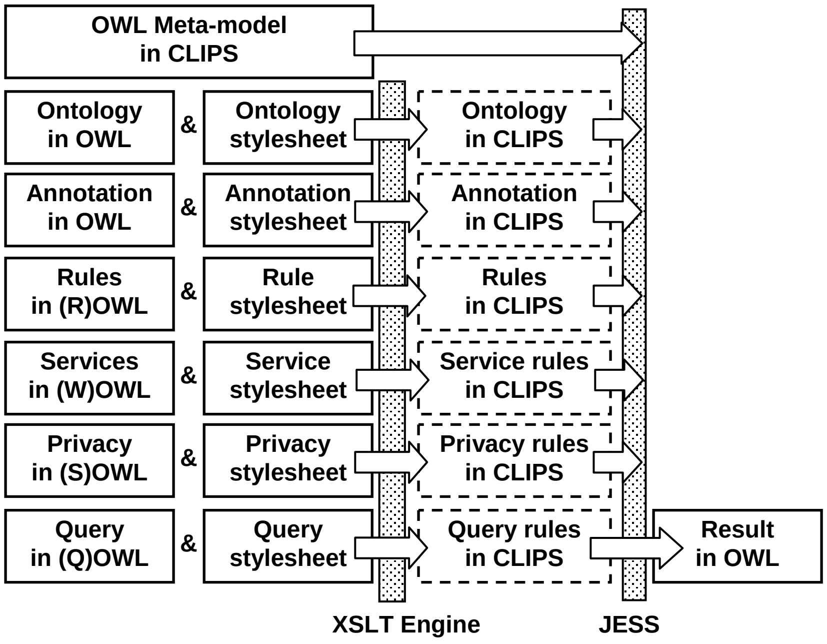


Figure 15. High-level overview of the Semantic e-Wallet

Knowledge is loaded into the e-Wallet by translating OWL input files into JESS assertions and rules, using a set of XSLT stylesheets. The OWL input files include ontologies and annotations that are transformed into (core) triple assertions, forward chaining rules (used to complete knowledge at the core layer) as well as service invocation rules and privacy enforcing rules – both represented as backward chaining rules. The XSLT templates act as meta-rules that generate the body, the head and typing (e.g. query transformation stylesheet in Figure 16).

```
(…)
<xsl:template match="/rdf:RDF">
 (defrule query (declare (salience 0))
  <xsl:for-each select="*[not(self::qowl:Query)]">
   <xsl:call-template name="process-class-instance"/>
  </xsl:for-each>
 =>
 (store-result<xsl:call-template name="variable-list"/>)
)
</xsl:template>

<xsl:template name="process-class-instance" >
(authorized_triple
 (predicate "&rdf;#type")
 (subject   <xsl:call-template name="local-ID">
             <xsl:with-param name="id">
              <xsl:value-of select="@rdf:about"/>
             </xsl:with-param>
            </xsl:call-template>)
 (object    "<xsl:value-of select="concat(namespace-uri(.),local-name(.))"/>")
)
 <xsl:for-each select="*">
  <xsl:call-template name="process-property-instance"/>
 </xsl:for-each>
</xsl:template>

<xsl:template name="process-property-instance" >
  <xsl:choose>
    <xsl:when test='count(*)=1'> <!-- has an element for child -->
      <xsl:call-template name="process-objectproperty-instance"/>
    </xsl:when>
    <xsl:when test='count(text())=1'> <!-- has an text for child -->
      <xsl:call-template name="process-dataproperty-instance"/>
    </xsl:when>
    <xsl:when test='@rdf:resource'> <!-- has a reference -->
      <xsl:call-template name="process-referenceproperty-instance"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>
(…)
```

Figure 16. Fragment of query transformation stylesheet

Once all this knowledge has been loaded and the forward chaining rules have been applied to complete the core knowledge base, the e-Wallet is ready to process incoming queries. A query is transformed into the need for an authorized triple. This in turn triggers privacy enforcing rules and generates needs for service triples. The service triples are generated by either migrating core triples or activating service invocation rules or a combination of both. This is further detailed below:

1. Queries have two components (see Figure 17): (a) an annotation about the query providing its context (e.g., who the sender of the query is), and (b) the query itself in the form of a pattern using a special namespace to identify variables. The context of a query is asserted for the time it takes to process it. We assume that security protocols (e.g., using digital signatures) are used to verify assertions about the query's context (e.g. verifying the identity of the sender). The query itself is transformed into a set of authorized triples in the privacy layer. These authorized triples form the body of a backward-chaining rule, whose head is a function that stores the results (in the form of variable instantiations) each time the rule is triggered and that generates RDF results in XML syntax ("pretty printing").
2. The need of the query for authorized triples triggers privacy enforcing rules. As illustrated in Figure 10, these rules have two roles. First, they check that the sender of the query has the required access rights. In addition, they also apply obfuscation rules to triples to ensure that the level of accuracy or inaccuracy provided in answers to queries is compatible with the user's privacy preferences. The need for authorized triples in combination with privacy enforcing rules generates a need for service triples.
3. The need for service triples in turn triggers service rules. First a generic service rule is applied that checks whether the needed service triple is not already available as a core triple. If this is the case an equivalent service triple is simply created. If there is no

equivalent core triple, the e-Wallet looks for matching rules that trigger internal function calls (e.g. getting the current time and date). If that fails too, it looks for matching (external) Web Services. To support this, we have extended the Jess library with internal functions (e.g. time) and functions to call external services.

```
<qowl:Query rdf:about="">
  <qowl:sender rdf:resource="http://cs.cmu.edu/~nsadeh"/>
</qowl:Query>
<mc:Person rdf:about="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://sadehlab.cs.cmu.edu/Variable#location" />
</mc:Person>
```

Figure 17. Query issued by the user 'nsadeh' requesting the location of user 'fgandon'

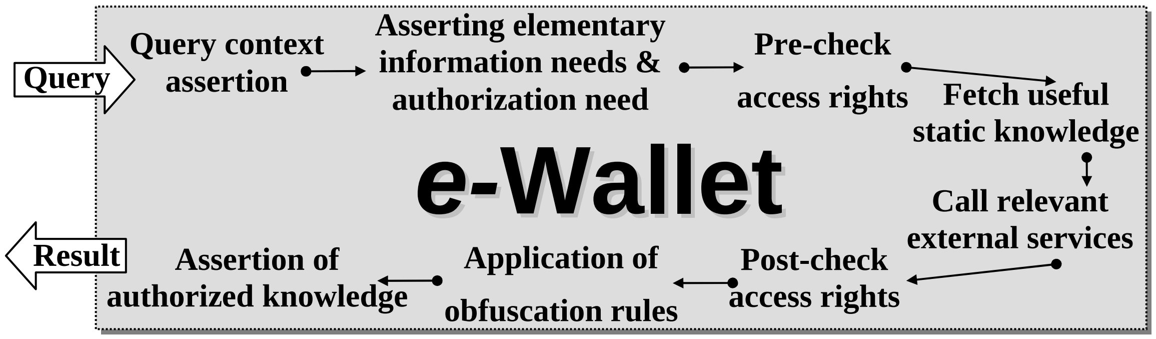
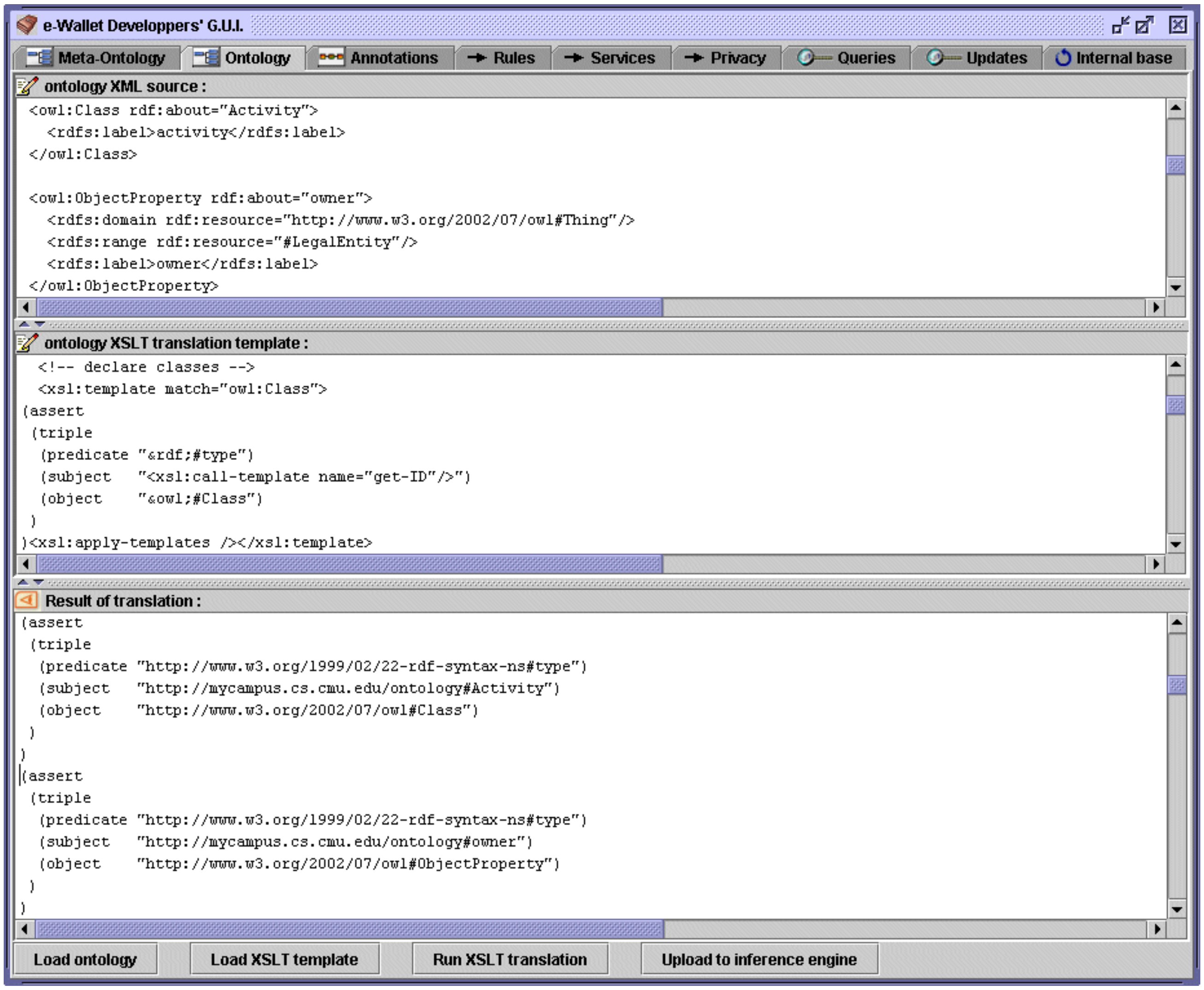

Query
Query context assertion
Asserting elementary information needs & authorization need
Pre-check access rights
Fetch useful static knowledge
e-Wallet
Call relevant external services
Post-check access rights
Application of obfuscation rules
Assertion of authorized knowledge
Result

Figure 18. Main steps involved in processing a query submitted to an eWallet.

An illustration of the main steps involved in processing a query submitted to a Semantic e-Wallet is provided in Figure 18.



ontology XML source :

```
<owl:Class rdf:about="Activity">
  <rdfs:label>activity</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:about="owner">
  <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:range rdf:resource="#LegalEntity"/>
  <rdfs:label>owner</rdfs:label>
</owl:ObjectProperty>
```

ontology XSLT translation template :

```
  <!-- declare classes -->
  <xsl:template match="owl:Class">
(assert
 (triple
  (predicate "&rdf;#type")
  (subject   "<xsl:call-template name="get-ID"/>")
  (object    "&owl;#Class")
 )
)<xsl:apply-templates /></xsl:template>
```

Result of translation :

```
(assert
 (triple
  (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
  (subject   "http://mycampus.cs.cmu.edu/ontology#Activity")
  (object    "http://www.w3.org/2002/07/owl#Class")
 )
)
(assert
 (triple
  (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
  (subject   "http://mycampus.cs.cmu.edu/ontology#owner")
  (object    "http://www.w3.org/2002/07/owl#ObjectProperty")
 )
)
```

Figure 19. Standalone e-Wallet development environment, as released on SemWebCentral.

A standalone version of the e-Wallet has been released on SemWebCentral (http://www.semwebcentral.org/). It includes a development environment that helps user debug ontologies, service invocation rules as well as privacy and security rules. The environment includes support for rollback points, enabling users to generate new facts when processing a query and later revert to a state that no longer includes those facts (e.g. to be ready to process the next query). A snapshot of the e-Wallet development environment is provided in Figure 19.

4 Results and Discussion

In contractual terms, this project was organized along the following tasks:

Task 1: Design, implement and evaluate Semantic Web functionality for the automated discovery of and access to personal resources: This task has been successfully completed through the development and evaluation of our Semantic e-Wallet as detailed in Section 3 (see also Appendix A for additional details)

Task 2: Design, implement and evaluate Semantic Web functionality for the automated discovery of and access to Web services: This task has been successfully completed through the development and evaluation of the MyCampus Semantic Web infrastructure for context-aware service provisioning, as detailed in Section 3.1 (see also appendix A for additional details)

Task 3: Design, Implement and Evaluate Functionality to Integrate AI planning and automated service discovery and access (Task 3):

(Subtask 3.1): Demonstrate and evaluate functionality integrating AI planning and automated Web service discovery. This subtask has been successfully completed, as outlined in Section 3.1 and as further detailed in [SSG04].

(Subtask 3.2): This task has only been completed at 50% due to a funding cut in Year 3.

Task 4: Provide regular contributions to the DAML Experiment (Task 4):

This task has been completed through contributions to the SONAT and CoSAR-TS demonstrations.

Task 5: Meetings and Reports (Task 5):

This task has been completed through participation in semi-annual DAML meetings, submission of quarterly reports and submission of the present final report to sponsor.

The following summarizes key findings and results of our project.

Key findings include:

- Semantic web technologies can play a key role **in facilitating the rapid development and maintenance of context-aware applications**, as demonstrated by the project's experience in several application domains (e.g. MyCampus domain aimed at enhancing everyday campus life through an open collection of context-aware applications, SONAT context-aware message filtering and delivery application, context-aware museum tour guide application)
- When extended to support **rules,** semantic web technologies provide a particularly powerful framework to capture and enforce **rich security and privacy policies** (including **context-sensitive policies**). Our efforts to develop general-purpose editors for such a powerful framework have shown however that there are significant usability issues that remain to be addressed. It is easy however to build special purpose editors that take advantage of the generic infrastructure we have developed

Additional accomplishments include:

- **Extensions of the OWL language** to represent rules and development of associated reasoning functionality
- **Public release of two software tools** (ROWL engine and Semantic eWallet) on SemWebCentral (http://www.semwebcentral.org/) – about 1,000 page views and 50 downloads in the first three months
- Demonstrations and evaluation on **both civilian and DoD-oriented scenarios** (e.g. MyCampus, SONAT, CoSAR-TS)
- Contributions to **semantic web service architecture (SWSA)** working group and **semantic web rules working groups**
- Development of a **semantic web architecture for context-aware service provisioning and privacy** that is influencing a number of ongoing R&D efforts in the telecommunications industry (e.g. work at Motorola, Nokia, NTTDoCoMo, Siemens, etc.). Extensions of this framework to incorporate AI planning functionality.
- Other significant accomplishments include:
    - Over a dozen publications and a best selling book on Mobile Commerce that discusses the relevance of Semantic Web technologies,
    - Multiple tutorials, keynotes, panels and courses
    - Posters and demonstrations at all DAML PI meetings during the course of the project
    - Member of the Joint US/EU Ad-Hoc Agent Markup Language Committee (“Joint Committee)
    - 2005 IBM Best Privacy Faculty Award for work on using Semantic Web technologies in support of privacy policies (co-recipient with Lorrie Cranor and Alessandro Acquisti) – selected out of about 20 nominations.

5 Conclusions

In conclusion, we believe that our project has made substantial contributions to the early development of Semantic Web technologies, helping advance the state of the art through extensions of languages and reasoning technologies and through early validations in the context of both civilian and DoD-relevant scenarios. Our work has produced several tools that we released on SemWebCentral. It has influenced ongoing standardization work (e.g. semantic web services and rules) and has produced an architecture for context-aware service provisioning and privacy that is influencing ongoing research efforts in industry. We believe that our work on interleaving Semantic Web reasoning and service discovery to enforce rich, context-sensitive security and privacy policies has the potential to provide for significantly more flexible approaches to decentralized trust management than what is currently available [SR05a]. While in the past reasoning technologies such as the ones embedded in Jess were considered too slow to enforce security policies, we believe that we have now reached a tipping point, where it is possible to enforce significantly richer policies and rely on significantly less scripted approaches to enforcing these policies. As both civilian and DoD organizations aim for increasingly tighter integration of their systems with those of others, demand for such technologies is bound to rapidly grow.

6 Recommendations

From a technical standpoint, our work as well as those of others in the DAML program (and beyond) has clearly demonstrated the far-reaching potential of the Semantic Web. It has also helped us better appreciate how much work still remains to be done before this potential can be fully harvested. This is not to say that existing languages and technologies can not already be put to good use – there are a number of ongoing efforts that aim at doing just that. There are however many deep technical challenges that still require sustained research investments in this area. From the standpoint of our own particular project, these challenges include issues of expressiveness, scalability and usability involved in capturing and enforcing rich semantic web service policies and the continued exploration of Semantic Web technologies for security and privacy (e.g. computation and communication tradeoffs associated with different policy configurations, development of meta-control strategies to integrate semantic web reasoning, service discovery and access, reasoning about conflicts of interest between entities seeking to gain access to resources and entities involved in helping build proofs to grant access, detection of deadlocks, real-time reasoning issues that require differentiating between classical negation and negation as failure, etc.).

Along with these research challenges, the Semantic Web needs more tools and early applications. Success will also depend on sustained commitment to incrementally reconciling research advances with ongoing industry-led standardization efforts.

Beyond our own particular project, looking at the DAML initiative as a whole, the past several years of funding have played a pivotal role in the early development of the Semantic Web. DAML helped create the necessary momentum (both domestically and internationally) for early standardization efforts and for the development of early demonstrators and tools. Over the past several years, the Semantic Web has grown from a somewhat exotic concept into an already rich collection of technologies and applications that is generating increasing interest among prospective users in both industry and government. It is a pity that, with prospective end-users' expectations so high, government support for the technology would seem to be dwindling. This situation is even more worrisome in light of the very substantial investments being made in this area elsewhere in the world (e.g. European IST program). While such considerations may not be at the top of the DoD's agenda, they will likely determine which part(s) of the world lead in what should ultimately be viewed as a strategic area not just in economical terms (e.g. e-services, enterprise integration, etc.) but also in terms of national security (e.g. intelligence, inter-organizational integration, coalition forces, cyber-security).

7 Selection of Project References

[CHGS05] S-C Chou, W-T Hsieh, F. Gandon and N. Sadeh, "Semantic Web Technologies for Context-Aware Museum Tour Guide Applications", *2005 International Workshop on Web and Mobile Information Systems (WAMIS'05)*, IEEE Computer Society.

[GS03] F. Gandon, and N. Sadeh. "A semantic e-wallet to reconcile privacy and context awareness". In *Proceedings of the Second International Semantic Web Conference (ISWC03)*, Florida, October 2003.

[GS03b] F. Gandon, and N. Sadeh. "Semantic web technologies to reconcile privacy and context awareness". Computer Science Technical Report CMU-CS-03-211, Dept. of Computer Science, Carnegie Mellon University, December 2003. Also available as CMU-ISRI-03-107.

[GS04a] F. Gandon, and N. Sadeh. "Semantic web technologies to reconcile privacy and context awareness". *Web Semantics Journal*, 1(3), 2004.

[GS04b] F. Gandon, and N. Sadeh. "ROWL: Rule language in OWL and translation engine for JESS". Mobile Commerce Laboratory, Carnegie Mellon University, 2004. http://mycampus.sadehlab.cs.cmu.edu/public_pages/ROWL/ROWL.html Also available on SemWebCentral.

[MJH+04] N. Miller, G. Judd, U. Hengartner, F. Gandon, P. Steenkiste, I. H. Meng, M. W. Feng, and Norman Sadeh. "Context-aware computing using a shared contextual information service". "Hot Spots", Pervasive 2004, April 2004, Vienna, *Advances in Pervasive Computing*, Austrian Computer Society (OCG), ISBN 3-85403-176-9, edited by Alois Ferscha, Horst Hoertner and Gabriele Kotsis.

[Sad02] Norman Sadeh, "M-Commerce: Technologies, Services and Business Models", Wiley, April, 2002

[Sad03] N. Sadeh. "MyCampus: Enhancing Everyday Campus Life". Video. October 2003. http://www-2.cs.cmu.edu/~sadeh/videos/mycampus_smaller.wmv

[SCV+03] N. M. Sadeh, T.C. Chan, L. Van, O. Kwon, and K. Takizawa. "Creating an open agent environment for context-aware m-commerce". *In Agentcities: Challenges in Open Agent Environments*, ed. by Burg, Dale, Finin, Nakashima, Padgham, Sierra, and Willmott, LNAI, Springer Verlag, pp.152-158, 2003.

[SG05] N. Sadeh and F. Gandon. "Ambient Intelligence: The MyCampus Experience". Submitted for publication as book chapter in "Ambient Intelligence, Wireless Networking, Ubiquitous Computing". Artech House, 2005.

[SR05a] N.M. Sadeh and J. Rao, "Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies", Submitted to 2005 AAAI Fall Symposium on Agents and the Semantic Web, May 2005. Longer version available as Carnegie Mellon University's School of Computer Science technical report CMU-ISRI-05-113.

[SSG04] M. Sheshagiri, N. Sadeh and F. Gandon, "Using Semantic Web Services for Context-Aware Mobile Applications", MobiSys 2004 Workshop on Context Awareness, Boston, June 2004.

8 Additional References

[APM04] R. Ashri, T. Payne, D. Marvin, M. Surridge and S. Taylor, Towards a Semantic Web Security Infrastructure. In Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, Stanford University, Stanford California.

[BSF02] Lujo Bauer, Michael A. Schneider and Edward W. Felten. "A General and Flexible Access Control System for the Web", In Proceedings of the 11th USENIX Security Symposium, August 2002.

[BFL96] Matt Blaze, Joan Feigenbaum, an Jack Lacy. “Decentralized Trust Management”. Proc. IEEE Conference on Security and Privacy. Oakland, CA. May 1996

[Coop93] Cooper, Jonathan. “Engaging the [Museum] Visitor: Relevance, Participation & Motivation in Hypermedia Design”. International Conference on Hypermedia and Interactivity in Museums (ICHIM), Cambridge, England, 1993. http://www.artgallery.nsw.gov.au/staff/jcooper/museum_education/engaging

[CLM+02] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler Marshall, and J. Reagle. The platform for privacy preferences 1.0 (P3P1.0) Specification. W3C Recommendation, April 16, 2002.

[CZC02] Corby, O., Faron-Zucker, C., Corese: A Corporate Semantic Web Engine, Workshop on Real World RDF and Semantic Web Applications, WWW Conference, Hawaii, (2002)

[DA99] Dey, A.K., Abowd, G.D.: Toward a Better Understanding of Context and Context-Awareness, GVU Technical Report GIT-GVU-99-22. College of Computing, Georgia Institute of Technology, (1999)

[DAML] DAML Joint Committee: DAML+OIL language, 27 March 2001, http://www.daml.org/2001/03/daml+oil-index.html

[DAMLS] DAML Services Coalition: DAML-S: Web Service Description for the Semantic Web, First International Semantic Web Conference, ISWC’02, Sardinia, Italy, LNCS 2342, (2002) 348-363

[Der99] Dertouzos, M.: The Future of Computing, Scientific American, August (1999)

[DKF+05] Li Ding, Pranam Kolari , Tim Finin , Anupam Joshi, Yun Peng and Yelena Yesha. "On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework", In Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security, March 2005.

[DSFA00] Dey, A., Salber, D., Futakawa, M., Abowd, G.: An Architecture to Support Context Aware Computing, Technical Report GIT-GVU-99-23. College Computing, Georgia Institute of Technology, Nov. (2000)

[EPAL] Schunter, M., Powers, C., The Enterprise Privacy Authorization Language (EPAL 1.1), IBM Research Laboratory, http://www.zurich.ibm.com/security/enterprise-privacy/epal/

[FIPA] FIPA, Specifications (2002) http://www.fipa.org/repository/fipa2000.html

[FN71] Fikes, R. E. and Nilsson, N. J., STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3-4): pages 189-208, 1971.

[Fri03] Friedman-Hill, E.: Jess in Action: Java Rule-based Systems, Manning Publications Com-pany, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/

[GPH03] Golbeck, J.; Parsia, B.; and Hendler, J. 2003. Trust networks on the Semantic Web. In Proceedings of 7th International Workshop on Cooperative Intelligent Agents, CIA 2003.

[GMM03] Guha, R., McCool, R., Miller E.: Semantic Search, Proceedings of WWW Conference, Budapest pp. 700-709 (2003)

[Gru93] T. R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993

[GSSS02] Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste P.: Project Aura: Towards Distraction-Free Pervasive Computing, IEEE Pervasive Computing, Special Issue on Integrated Perva-sive Computing Environments, Vol. 1, Number 2, April-June (2002) 22-31

[Hen01] Hendler, J.: Agents on the Web, IEEE Intelligent Systems, Special Issue on the Semantic Web, Vol. 16, No. 2, pp. 30-37, March/April, (2001)

[Hill03] Friedman-Hill, E.: Jess in Action: Java Rule-based Systems, Manning Publications Com-pany, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/

[HKL+04] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. Enabling context-aware and privacy-conscious user data sharing. In Proceedings of 2004 IEEE International Conference on Mobile Data Management, Berkeley, California, January 2004.

[HL01] Hong, J., Llanday, J.: A Context/Communication Information Agent, in Personal and Ubiquitous Computing Special Issue Situated Interaction and Context-Aware Computing, Vol. 5(1) (2001) 78-81

[HS04] U. Hengartner, and P, Steenkiste. Implementing access control to people location information. In 9th ACM Symposium on Access Control Models and Technologies (SACMAT'04), Yorktown Heights, June 2004

[HSSK04] T. van der Horst, T. Sundelin, K. E. Seamons, and C. D. Knutson. Mobile Trust Negotiation: Authentication and Authorization in Dynamic Mobile Networks. Eighth IFIP Conference on Communications and Multimedia Security, Lake Windermere, England, September 2004

[JESS] Friedman-Hill, E.: Jess in Action: Java Rule-based Systems, Manning Publications Company, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/

[KFJ03] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In Collection of IEEE 4th International Workshop on Policies for Distributed Systems and Networks, June 2003

[KPS04] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin and K. Sycara, Authorization and Privacy for Semantic Web Services, In Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, Stanford University, California, March 2004.

[LH03] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 331-339. ACM, 2003.

[LHL01] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Scientific American, May (2001)

[LGC+05] L.Bauer, S. Garriss, J. McCune, M.K. Reiter, J. Rouse, and P Rutenbar, "Device-Enabled Authorization in the Grey System", Submitted to USENIX Security 2005. Also available as Technical Report CMU-CS-05-111, Computer Science Department, Carnegie Mellon University, February 2005.

[LNOS04] T. Leithead, W. Nejdl, D. Olmedilla, K. Seamons, M. Winslett, T. Yu, and C. Zhang, How to Exploit Ontologies in Trust Negotiation. Workshop on Trust, Security, and Reputation on the Semantic Web, part of the Third International Semantic Web Conference, Hiroshima, Japan, November 2004.

[Mobi] MobiLife Project Website, http://www.ist-mobilife.org/, 2005.

[NAICS] North American Industry Classification System , http://www.census.gov/epcd/www/naics.html

[OS00] Opermann, R., and Specht, M., A Context-sensitive Nomadic Information System as an Exhibition Guide. Proc. of 2nd Intl. Symposium on Handheld and Ubiquitous Computing, Bristol, 2000.

[OWL] W3C: OWL Web Ontology Language Reference, Working Draft 31 March 2003, http://www.w3.org/TR/owl-ref/

[P3P] W3C: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, Recommendation 16 April 2002, http://www.w3.org/TR/P3P/

[PKPS02] Paolucci, M., Kawamura, T., Payne, T., and Sycara, K.: Semantic Matching of Web Service Capabilities, Proceedings of the 1st International Semantic Web Conference, Eds. I. Horrocks and J. Hendler, LCNS 2342, Springer Verlag, 2002.

[Rao04] J. Rao. "Semantic Web Service Composition via Logic-based Program Synthesis". *PhD Thesis*. Department of Computer and Information Science, Norwegian University of Science and Technology, December 10, 2004.

[RDF] W3C: RDF Vocabulary Description Language 1.0: RDF Schema, Working Draft 23 January (2003) http://www.w3.org/TR/rdf-schema/

[RKM04a] J. Rao, P. Küngas and M. Matskin, "Composition of Semantic Web Services using Linear Logic Theorem Proving". *To appear in Information Systems Journal - Special Issue on the Semantic Web and Web Services"*.

[RuleML] Boley, H., Grosof, B., Tabet, S., Wagner, G.: RuleML DTDs, The Rule Markup Initiative RuleML, http://www.dfki.uni-kl.de/ruleml/indtd0.8.html

[SAML] OASIS: Security Assertion Markup Language (SAML), Technology Reports, April 14 (2003) http://xml.coverpages.org/saml.html

[SAW94] Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Proc. of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, (1994) 85-90

[Schil94] Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Proc. of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, (1994) 85-90

[Schil95] Schilit, W.: A System Architecture for Context-Aware Mobile Computing, Ph.D. Thesis, Columbia University, 1995.

[SEH02] J. O'Sullivan, D. Edmond, and A. T. Hofstede. What's in a service? Towards accurate description of non-functional service properties. Distributedand Parallel Databases, 12:117.133, 2002.

[She04] Mithun Sheshagiri, Automatic Service Composition and Invocation for Semantic Web Services, University of Maryland, Baltimore County, May 2004.

[SJF03] Mithun Sheshagiri, Marie desJardins, and Timothy Finin, A Planner for Composing Services described in DAML-S, Workshop on Planning for Web Services, Trento, 2003

[UBJ04] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton and S. Aitken, Policy and Contract Management for Semantic Web Services. In Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series, Stanford University, Stanford California.

[UDDI] OASIS: Universal Description, Discovery and Integration standard, http://www.uddi.org/

[UPC+03] J. Undercoffer, F. Perich, A .Cedilnik, L. Kagal, and A. Joshi. A secure infrastructure for service discovery and access in pervasive computing. ACM Monet: Special Issue on Security in Mobile Computing Environments, October 2003

[OWLS] OWL-S: Semantic Markup for Web Services, W3C Submission Member Submission, November 2004. http://www.w3.org/Submission/OWL-S

[WHFG92] Want, R., Hopper, A., Falcao, V., Gibbons, J.: The Active Badge Location System, ACM Transactions on Information Systems 10(1) (1992) 91-102.

[WSDL] W3C: Web Services Description Language (WSDL) 1.1, Note 15 March 2001 http://www.w3.org/TR/wsdl

[WSH+03] Dan Wu, Evren Sirin, James Hendler, Dana Nau & Bijan Parsia, Automatic Web Service Composition Using SHOP2, Workshop on Planning for Web Services, Trento, 2003

[XACML] OASIS: Extensible Access Control Markup Language (XACML), Technology Reports, March 28 (2003) http://xml.coverpages.org/xacml.html

[XSLT] W3C: XSL Transformations (XSLT) Version 1.0, Recommendation 16 November 1999, http://www.w3.org/TR/xslt

9 Appendix A: A More In-Depth Overview of the Semantic e-Wallet

This appendix is based on a paper published in April 2004 in the Web Semantics Journal [GS04a].

9.1 Overall Introduction

Increasingly, application developers are looking for ways to provide users with added levels of convenience and ease of use through functionality that is capable of capturing the context within which they operate. This may involve knowing where the user is located, the task she is currently engaged in, her eating preferences, who her colleagues are as well as a variety of other contextual attributes. While there are many sources of contextual information, they tend to vary from one user to another and also over time. Different users may rely on different location tracking functionality provided by different cell phone operators; they may use different calendar systems, *etc.* Traditionally, context-aware applications and services have been hardwired to predefined sources of contextual information (e.g. relying on a particular set of sensors and protocols to track a user's locations). As a result, they remain prohibitively expensive to build and maintain and are few and between. We argue that what is needed is a more open environment, where context-aware applications can automatically discover and access a user's personal resources such as her calendar or location tracking functionality. This can be done by viewing each source of contextual information (or personal resource) as a Web service. Unfortunately, current Web Services standards such as UDDI [UDDI] or WSDL [WSDL] are not sufficient when it comes to describing a user's personal resources and to enabling automated access to them by context-aware applications. Another challenge, as we move towards more open platforms for access to a user's personal information, revolves around privacy issues. Users should be able to retain control over who has access to their personal information under different conditions. For instance, I may be willing to let my colleagues see where I am or access my calendar activities between 8am and 5pm on weekdays but not over the weekend. In addition, I may want to fine tune the granularity of the answer provided to a given query, depending on the context of that query. For instance, I may be willing to disclose the room that I am in to some people but only the city where I am to others In fact, I may even want to give different answers to different people, telling my secretary I am off to see my dentist, while telling my customers I am busy in a meeting.

In this paper, we introduce a Semantic Web architecture aimed at supporting the automated discovery and access of personal resources in support of a variety of context-aware applications. Within this architecture, each source of contextual information (e.g. a calendar, location tracking functionality, collections of relevant user preferences, organizational databases) is represented as a Semantic Web service. A central element of our architecture is its *semantic e-Wallet*, which acts as a directory of contextual resources for a given user, while enforcing her *privacy preferences*. Privacy preferences enable users to specify what information can be provided to whom in different contexts. They also allow users to specify what we call *obfuscation rules*, namely rules that control the accuracy or inaccuracy of the information provided in response to different queries under different conditions.

We have validated our architecture in the context of *my*Campus, a context-aware environment aimed at enhancing everyday campus life at Carnegie Mellon University (CMU). The environment revolves around a growing collection of task-specific agents capable of automatically accessing a variety of contextual information about their users (e.g. context-aware restaurant concierge, context-aware message filtering agent, *etc*.). This includes accessing their locations, calendar activities as well as a variety of other attributes and preferences. Students access the environment from PDAs over the campus's 802.11 wireless LAN. Empirical results obtained with a group of students over a period of several days are briefly summarized at the end of this article. While, in this paper, we focus on scenarios involving individual users, it should be noted that our architecture extends to scenarios where users are entire organizations. In this context, both organizations and individual users could each have one or more Semantic e-Wallets capable of leveraging a variety of individual or organizational knowledge subject to a rich set of privacy/confidentiality constraints.

The remainder of this article is organized as follows. Section 9.2 provides a brief overview of the state of the art in context-awareness, privacy, Web Services and the Semantic Web, emphasizing limitations of the work reported so far in the literature. In Section 9.3, we provide an overview of our Semantic Web environment for context-awareness and privacy. Section 9.4 focuses more specifically on the Semantic e-Wallet and includes a high-level scenario outlining its operation in response to a query about the current location of a user. Section 9.5 introduces the three layers of knowledge required to support the e-Wallet functionality. Section 9.6 discusses the e-Wallet's current implementation, which is based on OWL Lite [OWL], XSLT transformations [XSLT], and JESS [JESS]. Sections 9.7 and 9.8 provide further details on the e-Wallet's three layers. Section 9.9 discusses the interfaces to the e-Wallet and section 9.10 briefly describes some of our experiments carried out in our *my*Campus environment. Finally, section 9.11 summarizes what we view as the main contributions of our work along with some concluding remarks.

9.2 Prior Work

Prior efforts to develop context aware applications are many. Early work in context awareness includes the Active Badge System developed at Olivetti Research Lab to redirect phone calls based on people's locations [WHFG92]. The ParcTab system developed at the Xerox Palo Alto Research Center in the early nineties relied on PDAs to support a variety of context-aware office applications (e.g. locating nearby resources such as printers, posting electronic notes in a room, *etc*.) [Schil94, Schil95]. Other relevant applications that have emerged over the years range from location-aware tour guides to context-aware memory aids. More recent research efforts in context awareness include MIT's Oxygen [Der99], CMU's Aura [GSSS02] and several projects at Berkeley's GUIR (e.g. [HL01]) to name just a few.

While early context-aware applications relied on *ad hoc* architectures and representations, it was quickly recognized that separating the process of acquiring contextual information from actual context-aware applications was key to facilitating application development and maintenance. Georgia Tech's Context Toolkit represents the most significant effort in this direction [DA99, DSFA00]. In the Context Toolkit, widgets act as wrappers that provide access to different sets of contextual information (e.g. user location, identity, time and activity), while insulating applications from context acquisition concerns. Each user (as well as other relevant entities such as physical objects or locations) has a context server that contains all the widgets relevant to it. This is similar to our notion of e-Wallet, which serves as a directory of all personal resources relevant to a given user (e.g. relevant location tracking functionality, relevant collections of preferences, access to one or more calendar systems, *etc*.). Our Semantic e-Wallet however goes one step beyond Dey's Context Toolkit. It makes it possible to leverage much richer models of personal resources - what personal information they give access to, when to access one rather than the other, how to go about accessing these resources. In addition, it includes access control and obfuscation functionality to enforce user privacy preferences. This richer model is key to supporting automated discovery and access of a user's personal resources by agents. In other words, while the Context Toolkit focuses mainly on facilitating the development of context-aware applications through off-line, re-use and integration of context-aware components (i.e. widgets), our architecture emphasizes real-time, on-the-fly queries of personal resources by context-aware agents. These queries are processed through several layers of functionality that support automated discovery and access of relevant personal resources subject to user-specified privacy preferences.

The notion of e-Wallet as introduced in systems such as Microsoft's .NET Passport is not new. However current implementations have been limited to storing a very small amount of information and offer very restricted control to the user when it comes to specifying what information can be made available to different services. For instance, in Passport, users can specify whether or not they are willing to share parts of their profiles with all participating sites but cannot distinguish between different participating sites. Our notion of Semantic e-Wallet lifts these restrictions and allows users to control access to any of their personal resources. It also allows for multiple sources of similar information (e.g. multiple calendars or multiple location tracking functionality) and for functionality that can dynamically select which of these resources to tap based on the context and the nature of the query at hand (e.g. using your car's GPS system when you are driving and your cell phone operator's location tracking functionality when you are not).

Our notion of semantic e-Wallet extends recent efforts to develop rich languages for capturing user privacy preferences such as P3P's APPEL language [P3P]. It does so by making it possible to leverage any number of domain ontologies and by allowing for preferences that relate to any number of contextual attributes. In addition, it allows users to specify obfuscation rules through which they control the level of accuracy (or inaccuracy) at which their contextual information is disclosed to different parties under different conditions. This includes telling some people which room you are in, while simply telling others whether you are at work or not, or whether you are in town or not. It also includes scenarios where you might want to pretend you are in one place, while you are really elsewhere.

Last but not least, while the security community has developed powerful languages to capture access control privileges such as the Security Assertion Markup Language

(SAML) [SAML], the XML Access Control Markup Language (XACML) [XACML] and the Enterprise Privacy Authorization Language (EPAL) [EPAL], these languages do not take advantage of Semantic Web concepts. Our work builds directly on recent efforts aimed at moving the Web from an environment where information is primarily made available for human consumption to one where it is annotated with semantic markup that makes it understandable to software applications. These efforts are part of a long-term vision generally referred to as the *Semantic Web* [LHL01, Hen01]. They have already resulted in a succession of semantic markup languages [DAML, OWL] as well as early efforts to define Web Service ontologies and markup in the context of languages such as DAML-S [DAMLS]. In our work, we have relied on the use of DAML+OIL [DAML] and more recently OWL [OWL] to represent contextual information (e.g. location, calendar activities, social and organizational relationships, *etc*.) and privacy preferences and on Semantic Web service concepts to support the automated discovery and access of personal and public resources.

9.3 Overall System Architecture

We consider an environment where, over time, users purchase (or subscribe to) different sets of task-specific agents. These agents are each intended to assist them in the context of different activities (e.g. scheduling meetings with colleagues, reminding them of purchases they need to make, arranging trips or filtering incoming messages). To function, each agent needs to access some information about its user as well as possibly other users. Access to a user's personal (or contextual) information is controlled by that user's e-Wallet subject to privacy (enforcing) rules. The e-Wallet Manager (or simply e-Wallet) serves as a repository of static knowledge about the user – just like .NET Passport, except that here knowledge is represented using OWL. In addition, the e-Wallet contains knowledge about how to access more information about the user by invoking a variety of resources, each represented as a Web Service. This knowledge is stored in the form of rules that map different contextual attributes onto one or more possible service invocations, enabling the e-Wallet to automatically identify and activate the most relevant resources in response to queries about the user's context (e.g. accessing the user's calendar to find out about her availability, or consulting one or more location tracking applications in an attempt to find out about her current location). User-specified privacy rules, also stored in the e-Wallet, ensure that information about the user is only disclosed to authorized parties, taking into account the context of the query. They further adjust the accuracy or inaccuracy of the information provided in accordance with the user's obfuscation preferences.

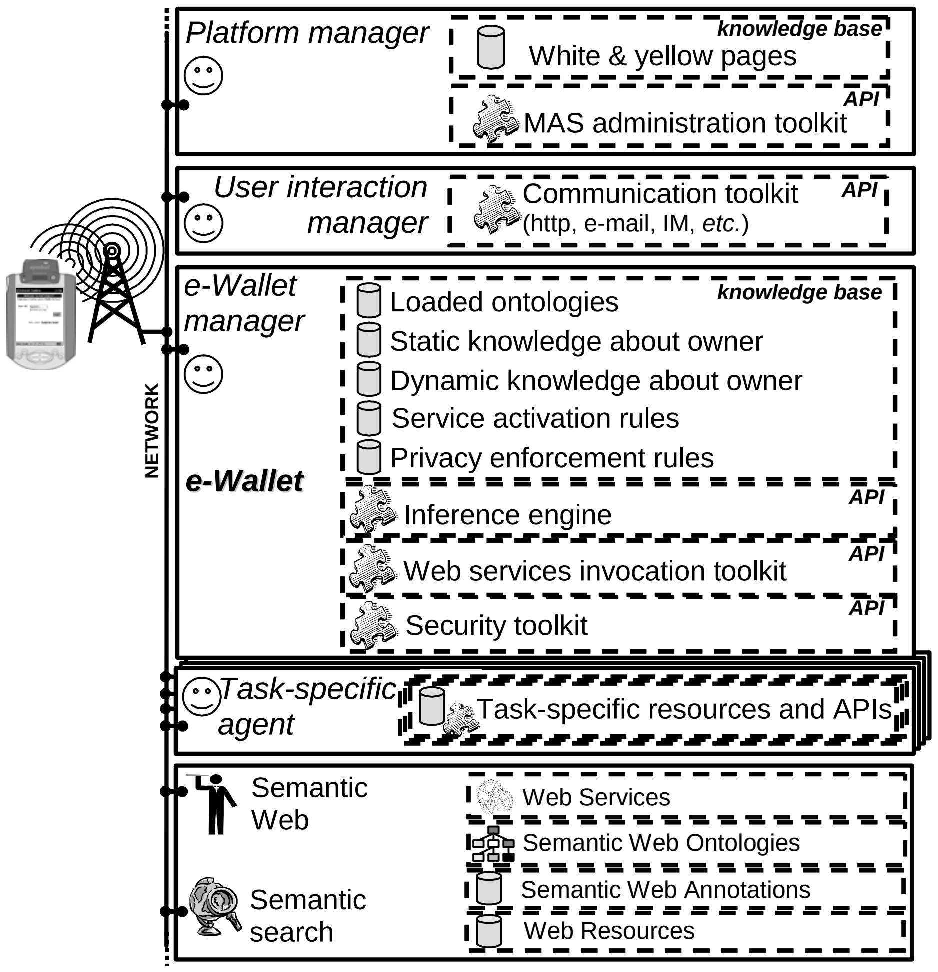


Figure 20. *my*Campus architecture: a user's perspective - the smiley faces represent agents.

Figure 20 provides an overview of our Semantic Web environment. It illustrates a situation where access is from a PDA over a wireless network, as is the case in *my*Campus, the environment in which we have instantiated our architecture. However, our architecture extends to fixed Internet scenarios and more generally to environments where users can connect to the infrastructure through a number of access channels and devices – information about the particular access device and channel can actually be treated as part of the user's context and be made available through her e-Wallet. As can be seen in Figure 1, other key elements of our architecture include:

- One or more *Platform Managers* that build on top of Directory Facilitators and Agent Management Systems, as defined in FIPA [FIFA]. They manage the agents running at their sites, and maintain white and yellow page directories of these agents and the services they provide.

- *User Interaction Managers* that are responsible for interactions with the user. This includes managing login sessions as well as interactions with the user's agents and her e-Wallet. Because different users interact with different sets of agents, this also includes the dynamic generation of interfaces for interacting with these agents and the customization of these interfaces to the current interaction context (e.g. particular access device). Communication with the User Interaction Manager typically takes place through a number of APIs, e.g. an Instant Messaging API, an HTTP/HTML API, *etc.*

Clearly, agents are not limited to accessing information about users in the environment. Instead, they also typically access public Web Services, Semantic Web annotations, public ontologies and other public resources. On CMU's campus, where we have deployed *my*Campus, this includes access to a variety of services such as 23 restaurant web services or a pubic weather forecasting web service.

In the following sections, we focus on the e-Wallet functionality. Additional details on *my*Campus and some of the agents we have deployed can be found in [SCV+03].

9.4 Semantic e-Wallet

The e-Wallet is a central element of our Semantic Web architecture for context-awareness and privacy. It provides a unified and secure semantic interface to all the user's personal resources, enabling agents in the system, whether working for the owner of the e-Wallet or for other users, to access and, when appropriate, modify information about the user subject to that user's privacy preferences (e.g. not just determining whether the user is available between 3 and 4pm but also, possibly, scheduling a meeting at that time). The e-Wallet is *not* a static information repository. While it does contain some static information about the user, it is an agent acting as clearinghouse and gatekeeper for a user's personal resources. Its knowledge about the user, her personal resources and preferences falls into four categories:

1. *Static knowledge*. This context-independent knowledge typically includes the user's name, her email address, employer, home address as well as context-independent preferences (e.g. "I like spicy vegetarian cuisine"). This knowledge, like all other in the e-Wallet, can be edited by the user via the User Interaction Manager.
2. *Dynamic knowledge*. This is context-sensitive knowledge about the user, often involving a variety of preferences such as "When driving, I don't want to receive instant messages".
3. *Service invocation rules*. These rules help leverage information resources external to the e-Wallet – both personal and public. They effectively turn the e-Wallet into a semantic directory of personal resources that can be automatically discovered and accessed to process incoming queries. Specifically, service invocation rules provide a mapping between contextual attributes and personal resources available to access these attributes, viewing each personal resource as a Semantic Web service. An example of one such mapping is a rule indicating that a query about the user's current activity can be answered by accessing her Microsoft Outlook calendar. We have developed Web Service wrappers for a variety of personal resources such as Microsoft Outlook Calendar or location tracking functionality. Service invocation rules are not limited to providing a one-to-one mapping between contextual attributes and personal resources. Instead, they can leverage rich ontologies of personal resources, enabling the e-Wallet to select among a number of possible personal resources based on availability,

accuracy and other relevant considerations. For instance, in response to a query about the user's location, the rules can specify that, when the user is driving, the best method available is the GPS in her car. If she is at work and her wireless-enabled PDA is on, her location can be obtained using location tracking functionality running over the enterprise's wireless LAN. If everything else fails, her calendar might have some information about her location. Finally, it should be noted that to answer queries about the user, additional mapping rules that support automated discovery and access of public services may also be needed. For instance, a query like "Tell me whether Fabien is in a sunny place right now" will typically require accessing Fabien's location as well as a public weather service.

4. *Privacy preferences*. These preferences encapsulate knowledge about what information about herself the user is willing to disclose to others under different conditions. These preferences themselves fall into two categories:
   - *Access control rules.* These rules simply express who has the right to see what information under different conditions e.g. "My location should only be visible to members of my team during week days between 8am and 5pm".
   - *Obfuscation rules.* Often user privacy preferences are not black-and-white but rather involve different levels of accuracy or inaccuracy: *Obfuscation by abstraction* is about abstracting away some details about the user's current context such as telling people whether or not you are in town without giving your exact location. *Obfuscation by falsification* is about scenarios where the user may not want to appear as if she is withholding information but would rather provide false information. For instance, a user may not want to reveal her true email address to a web service for fear of getting spammed.

All the above knowledge (including rules) is represented in OWL. It can leverage a number of relevant ontologies (e.g. ontologies about contextual attributes, personal resources, as well as more specific knowledge such as cuisine types and food preferences or message types and message filtering preferences).

Before delving deeper into the details of the e-Wallet, a scenario will help illustrate the key steps it goes through in processing incoming queries (Figure 21). For the sake of argument, we will assume a query submitted by a user (Norman) to the e-Wallet of a second user (Fabien) inquiring about that second user's current location. The main steps are as follows:

1. *Asserting the query's context*: As a first step, facts about the context of the query are asserted – namely they are loaded into the e-Wallet's inference engine for possible use as part of inferences to be made in processing the query. In our example, one such assertion is that "the sender of the query is Norman".
2. *Asserting elementary information needs and the need to go through an authorization process*: Here the query is translated into an aggregate goal that includes (a) a combination of elementary information needs – in our example the need to find "Fabien's location", along with (b) a requirement to go through an authorization process. The authorization process, which is distributed across some of the following steps, results in the request being either denied or cleared, the latter possibly following the application of obfuscation rules. In our example, the authorization goal requires checking that Norman is entitled to having access to Fabien's location and that the

level of resolution at which the query is answered is compatible with Fabien's privacy preferences.

3. *Pre-checking whether the query is allowable*: A first check is performed to see whether the query is allowable based on access rights considerations. In our example, the e-Wallet checks whether Norman is allowed to inquire about Fabien's location. Fabien's e-Wallet might include a privacy preference specifying that his colleagues at work can see the building that he is in, when he is on campus, but that no one else should be given access to his location. In this first check, the e-Wallet might be able to determine that Norman is indeed a colleague of Fabien's – e.g. based on organizational knowledge stored in the static knowledge base of Fabien's e-Wallet. At this stage, because it has not yet determined whether Fabien is on campus or not, the e-Wallet has no ground for denying the request. Therefore, it continues processing it, as detailed below.
4. *Checking the e-Wallet's local knowledge base*: Some queries can be answered in whole or in part, using facts in the e-Wallet's local knowledge base, which, as we have seen in Section 3, contains both static (namely, context-independent) and dynamic (namely, context-sensitive) knowledge about the user. In our particular example, such knowledge is not particularly helpful and the e-Wallet needs to turn to outside sources of personal information to answer the query (see next step).
5. *Invoking personal resources as Web services*: When local knowledge is not sufficient to answer a query, the e-Wallet turns to its service invocation rules to identify external resources that might help answer it. This may involve accessing one or more of the user's personal resources such as his calendar and/or one or more trusted public services. In our example, the campus where Fabien works has a wireless LAN that supports location tracking. This functionality can be invoked by the e-Wallet to obtain Fabien's location. The actual invocation takes place through the web service invocation toolkit already introduced in Figure 1.
6. *Post-checking whether the query is allowable*: armed with additional knowledge obtained by invoking one or more external resources, the e-Wallet is now in a better position to check whether the query is allowable. In our example, colleagues of Fabien's are only allowed to see his location when he is on campus. Assuming that Fabien is on campus, the request is now deemed allowable. This does not mean however that the authorization process required as part of the goals set in step 2 has been fully completed. Obfuscation rules may still need to be applied.
7. *Application of Obfuscation Rules*: suppose that the location tracking functionality used to answer our query about Fabien's location returned the specific room he is in, while Fabien is only willing to disclose the buildings that he is in. This latter requirement is captured by the e-Wallet in the form of an obfuscation rule that returns the building in which Fabien is rather than the exact room. Application of this rule will typically involve accessing ontologies about rooms and buildings as well as annotations about the campus where Fabien works.
8. The query has now been fully processed and an acceptable answer generated. This answer (e.g. "Fabien is in Smith Hall") can be returned to Norman.

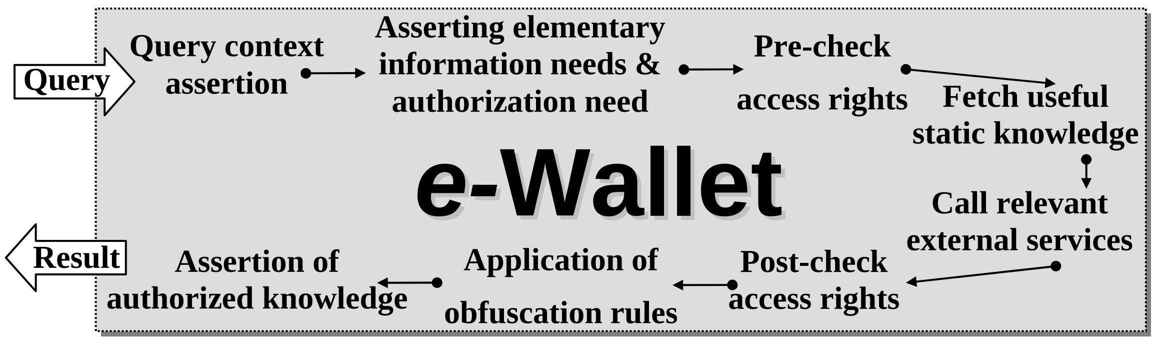

Query
Query context assertion
Asserting elementary information needs & authorization need
Pre-check access rights
Fetch useful static knowledge
e-Wallet
Call relevant external services
Post-check access rights
Application of obfuscation rules
Assertion of authorized knowledge
Result

Figure 21. Main steps involved in processing a query submitted to an e-Wallet.

9.5 A Three-Layer e-Wallet Implementation

As shown in Figure 22, we developed a three-layer implementation of our e-Wallet:

- *Core Layer*: At the most basic level, the e-Wallet's knowledge includes an OWL meta-model – required to interpret OWL statements. In addition, it maintains both static (context-independent) and dynamic (context-dependent) knowledge about the user. This knowledge is obtained by loading available annotations about the user along with relevant ontologies and is currently completed using forward-chaining reasoning – to avoid having to infer the same facts over and over again. Knowledge in this layer is represented using a (core) triple template:
  `(predicate, subject, object)`$_{\text{TRIPLE}}$

- The *Service Layer* completes the e-Wallet's core knowledge with invocation rules that map information retrieval goals about contextual attributes onto external service invocations. These are modeled as backward-chaining rules. Given an information retrieval goal such as "Give me Fabien's location", they help identify and invoke one or more relevant information resources, each modeled as a Web service, as already discussed in Section 4. Knowledge in this layer is represented using a special type of triple called "service triple" denoted:
  `(predicate, subject, object)`$_{\text{SERVICE TRIPLE}}$

  Service triples reside in the service layer and are created in either of two ways. They can result from the migration of a triple from the core layer or from the activation of an invocation rule (e.g. an assertion about Fabien's location as returned by a call to a location tracking service). Migration between the core layer and the service layer is implemented by rules specifying that any (core) triple can be used to generate an equivalent service triple.
- The outer layer is referred to as the *Privacy Layer*, as this is where privacy (enforcing) rules are applied. Assertions in this layer are represented as another special type of triple called "authorized triple":
  `(predicate, subject, object)`$_{\text{AUTHORIZED TRIPLE}}$

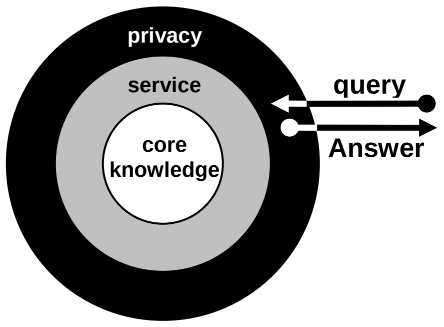


Figure 22. e-Wallet’s 3 layer implementation

Only authorized triples can be sent in response to queries. Authorized triples are generated by applying privacy enforcing rules to service triples, thereby ensuring that information about the user is only disclosed to authorized parties and in accordance with relevant obfuscation rules.

Privacy enforcing rules are encoded as backward-chaining rules. These rules map needs for authorized triples onto needs for service triples to be post-processed subject to the privacy enforcing rules. Upon receiving an incoming query, the e-Wallet generates a need for one or more authorized triples. This need in turn typically triggers needs for service triples and core triples, eventually resulting either (a) in the generation of authorized triples that can be returned in response to the query or (b) in an exception, if the query is found unallowable (e.g. an unauthorized party requesting your location or trying to schedule a meeting in your calendar). In summary, security in our architecture is directly enforced through typing.

9.6 Additional Implementation Considerations

The current implementation of our e-Wallet is based on JESS, a high-performance Java-based rule engine that supports both forward and backward chaining – the latter by reifying "needs for facts" as facts themselves, which in turn trigger forward-chaining rules. The e-Wallet’s knowledge base is initialized with: (a) a model of RDF [RDF] triples as a template for unordered facts, (b) a model of specialized triples used in our three layers (core triples, service triples and authorized triples) along with associated migration rules between the layers, and (c) an OWL meta-model.

Additional knowledge is loaded into the e-Wallet by translating OWL input files into JESS assertions and rules, using a set of XSLT stylesheets [XSLT] (figure 23). The OWL input files include ontologies and annotations that are transformed into (core) triple assertions, forward-chaining rules (used to complete knowledge at the core layer) as well as service invocation rules and privacy enforcing rules – both represented as backward-chaining rules. The XSLT templates act as meta-rules that generate the body, the head and typing used by the JESS rules (e.g. query transformation stylesheet in figure 24).

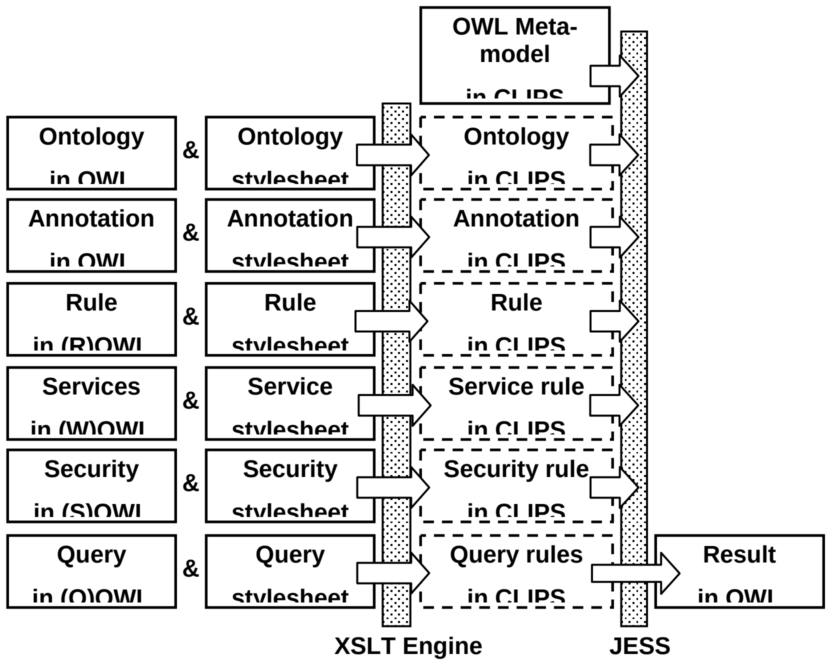


Figure 23. High-level flows and processes in the e-Wallet

```
(…)
<xsl:template match="/rdf:RDF">
 (defrule query (declare (salience 0))
  <xsl:for-each select="*[not(self::qowl:Query)]">
   <xsl:call-template name="process-class-instance"/>
  </xsl:for-each>
 =>
 (store-result<xsl:call-template name="variable-list"/>)
)
</xsl:template>

<xsl:template name="process-class-instance" >
(authorized_triple
 (predicate "&rdf;#type")
 (subject   <xsl:call-template name="local-ID">
             <xsl:with-param name="id">
              <xsl:value-of select="@rdf:about"/>
             </xsl:with-param>
            </xsl:call-template>)
 (object    "<xsl:value-of select="concat(namespace-uri(.),local-name(.))"/>")
)
 <xsl:for-each select="*">
  <xsl:call-template name="process-property-instance"/>
 </xsl:for-each>
</xsl:template>

<xsl:template name="process-property-instance" >
  <xsl:choose>
    <xsl:when test='count(*)=1'> <!-- has an element for child -->
      <xsl:call-template name="process-objectproperty-instance"/>
    </xsl:when>
    <xsl:when test='count(text())=1'> <!-- has an text for child -->
      <xsl:call-template name="process-dataproperty-instance"/>
    </xsl:when>
    <xsl:when test='@rdf:resource'> <!-- has a reference -->
      <xsl:call-template name="process-referenceproperty-instance"/>
    </xsl:when>
  </xsl:choose>
</xsl:template>
(…)
```

Figure 24. Fragment of the query transformation stylesheet

```
<qowl:Query rdf:ID="">
  <qowl:sender rdf:resource="http://cs.cmu.edu/~nsadeh"/>
</qowl:Query>
<mc:Person rdf:about="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://sadehlab.cs.cmu.edu/Variable#location" />
</mc:Person>
```

Figure 25. Query issued by the user 'nsadeh' requesting the location of user 'fgandon'

Once all this knowledge has been loaded and the forward-chaining rules have been applied to complete the core knowledge base, the e-Wallet is ready to process incoming queries. A query is transformed into the need for an authorized triple. This in turn triggers privacy enforcing rules and generates needs for service triples. The service triples are generated by either migrating core triples or activating service invocation rules or a combination of both. This is further detailed below:

1. Queries have two components (see Figure 25): (a) an annotation about the query providing its context (e.g., who the sender of the query is), and (b) the query itself in the form of a pattern using a special namespace to identify variables. The context of a query is asserted for the time it takes to process it – later, clean up rules take care of removing all assertions created while processing it. We assume that security protocols (e.g., using digital signatures) are used to verify assertions about the query's context (e.g. verifying the identity of the sender). The query itself is transformed into a set of authorized triples in the privacy layer. These authorized triples form the body of a backward-chaining rule, whose head is a function that stores the results each time the rule is triggered and that generates OWL results in XML syntax ("pretty printing").
2. The need of the query for authorized triples triggers privacy enforcing rules. As illustrated in Figure 27, these rules have two roles. First, they check that the sender of the query has the required access rights. In addition, they also apply obfuscation rules to triples to ensure that the level of accuracy or inaccuracy provided in answers to queries is compatible with the user's privacy preferences. The need for authorized triples in combination with privacy enforcing rules generates a need for service triples.
3. The need for service triples in turn triggers service rules. First a generic service rule is applied that checks whether the needed service triple is not already available as a core triple. If this is the case an equivalent service triple is simply created. If there is no equivalent core triple, the e-Wallet looks for matching rules that trigger internal function calls (e.g. getting the current time and date). If that fails too, it looks for matching (external) Web Services. To support this, we have extended the Jess library with internal functions (e.g. time) and functions to call external services. An example of a service invocation rule is given in Figure 28.

Figure 26 uses pseudo-code to depict the e-Wallet's overall processing flow.

```
//Load CLIPS model of RDF/S and OWL                        INITIALIZATION
{∀ triple($T_i$) | $T_i$ ∈ $OWL_{model}$ } assert ($T_i$) in JESS
{∀ rule($R_i$) | $R_i$ ∈ $OWL_{model}$ } defrule ($R_i$) in JESS
//Load ontologies
{∀ $O_{OWLi}$ | Ontology ($O_{OWLi}$)}
```

```
  $O_{CLIPSi}$:=Ontology stylesheet ($O_{OWLi}$) // extract ontology triples
  {$\forall$ triple($T_i$) | $T_i \in O_{CLIPSi}$ } assert($T_i$) in JESS
//Load annotations
{$\forall$ $A_{OWLi}$ | Annotation ($A_{OWLi}$)}
  $A_{CLIPSi}$:=Annotation stylesheet ($A_{OWLi}$) // extract annotation triples
  {$\forall$ triple($T_i$) | $T_i \in A_{CLIPSi}$ } assert($T_i$) in JESS
//Load rules
{$\forall$ $R_{ROWLi}$ | Rule ($R_{ROWLi}$)}
  $R_{CLIPSi}$:=Rule stylesheet ($R_{ROWLi}$) //makes forward rules producing triples
  {$\forall$ rule(Ri) | Ri $\in$ RCLIPSi } defrule (Ri) in JESS
//Load service rules
{$\forall$ $S_{WOWLi}$ | service description ($S_{WOWLi}$)}
  $S_{CLIPSi}$:=Service stylesheet ($S_{WOWLi}$)
                    //makes backward rules producing dynamic triples
  {$\forall$ rule(Ri) | Ri $\in$ SCLIPSi } defrule (Ri) in JESS
//Load privacy rules
{$\forall$ $P_{ROWLi}$ | Security rule ($P_{ROWLi}$)}
  $P_{CLIPSi}$:=Privacy stylesheet ($P_{POWLi}$)
                  // makes backward rules producing authorized triples
  {$\forall$ rule(Ri) | Ri $\in$ PCLIPSi } defrule (Ri) in JESS
------------------------------------------------------------------
                                                  QUERY PROCESSING
//Load query
$C_{CLIPSi}$:=Query stylesheet ($A_{OWLi}$) // extract context triples e.g.: sender
{$\forall$ triple($T_i$) | $T_i \in C_{CLIPSi}$ } assert($T_i$) in JESS
$Q_{CLIPSi}$ := Query stylesheet ($Q_{POWLi}$)
               // makes one backward rule requiring authorized triples
defrule ($Q_{CLIPSi}$) in JESS
(Run the RETE Algorithm)
```

Figure 26. e-Wallet's overall processing flow.

The system has been designed with efficiency in mind. For instance, query processing stops as soon as an authorization violation is detected. Also, at the service layer, rules ensure that the system first checks for available core triples before attempting to invoke external resources.

We currently use RDF-S/OWL to represent rules. In comparison to RuleML [RuleML], we do not reify the role of the relation and its arguments. We simply use triples to represent rules and take advantage of the typing mechanism of the XML syntax. We use a special namespace to identify variables. While we are following ongoing developments in RuleML [RuleML] and OWL Rules, our current focus is on the use of rules that apply to OWL assertions. Later we could easily extend our system, for instance using XSLT stylesheets to translate between our representation of rules in OWL and RuleML representations.

As shown in Figure 27, *privacy enforcing rules* are defined using three tags: the content of the *target tag* describes the piece of knowledge to which this rule applies; the content of the *check tag* describes the conditions under which read access is granted; the content of the *revision tag* describes the obfuscation to be applied before migrating triples to the authorized layer. Note that, at the time of writing, our e-Wallet also supports limited write access rules.

As shown in Figure 28 the *service rules* have three child tags: the content of the *output tag* describes the piece of knowledge that this rule can produce; the content of the *precondition tag* describes the knowledge needed for calling the service; the content of the *call tag* describes the function to trigger and its parameters. For reference, the CLIPS representation of this rule, following the application of our XSLT transformation, is also provided in Figure 29.

```
<sowl:ReadAccessRule>
  <rdfs:label>people can only know whether or not I am on campus</rdfs:label>
  <sowl:target>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="&variable;#location"/>
    </mc:Person>
  </sowl:target>
  <sowl:check>
   <rowl:And>
     <rowl:condition>
       <mc:E-Wallet rdf:about="&variable;#e-Wallet">
         <mc:owner>
          <mc:Person rdf:about="&variable;#owner"/>
         </mc:owner>
       </mc:E-Wallet>
     </rowl:condition>
     <rowl:condition>
       <mc:Place rdf:about="http://www.cmu.edu">
          <mc:include rdf:resource="&variable;#location" />
       </mc:Place>
     </rowl:condition>
     <rowl:not-condition>
       <qowl:Query rdf:about="&variable;#query">
         <qowl:sender rdf:resource="&variable;#owner" />
       </qowl:Query>
     </rowl:not-condition>
   </rowl:And>
  </sowl:check>
  <sowl:revision>
    <mc:Person rdf:about="&variable;#owner">
      <mc:location rdf:resource="http://www.cmu.edu"/>
    </mc:Person>
  </sowl:revision>
</sowl:ReadAccessRule>
```

Figure 27. Privacy rule obfuscating the location of the owner

```
<wowl:ServiceRule wowl:salience="50">
  <rdfs:label>provide activity status for a person</rdfs:label>
  <wowl:output>
    <mc:Person rdf:ID="&variable;#person">
      <mc:has_activity rdf:resource="&variable;#activity" />
    </mc:Person>
  </wowl:output>
  <wowl:precondition>
    <mc:Person rdf:ID="&variable;#owner">
      <mc:PDA_endpoint>&variable;#endpoint</mc:PDA_endpoint>
    </mc:Person>
  </wowl:precondition>
  <wowl:call>
   <wowl:Service wowl:name="call-web-service">
     <wowl:qname>http://mycampus/PDAService#</wowl:qname>
     <wowl:endpoint>&variable;#endpoint</wowl:endpoint>
     <wowl:method>GetCurrentWeekAppointments</wowl:method>
     <wowl:user_id>&variable;#owner</wowl:user_id>
   </wowl:Service>
  </wowl:call>
</wowl:ServiceRule>
```

Figure 28. Service rule for activity-tracking invocation in WOWL

```
(defrule provide_activity_status_for_a_person (declare (salience 50))
  (need-dynamic_triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#has_activity")
   (subject   ?person)
   (object    ?activity)
  )
  (dynamic_triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?owner)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
  (dynamic_triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#PDA_endpoint")
   (subject   ?owner)
   (object    ?endpoint)
  )
  =>
    (call-web-service qname "http://mycampus/PDAService#"
                      endpoint ?endpoint
                      method "GetCurrentWeekAppointments"
                      user_id ?owner)
)
```

Figure 29. Service rule for activity-tracking invocation translated in CLIPS

9.7 Static Knowledge and Domain Specific Rules

As indicated earlier, the RDF triple meta-model is defined as a template used in forward chaining rules. The OWL meta-model is asserted as a list of unordered facts such as the one shown in Figure 30. The semantics attached to properties is translated into rules as illustrated in Figure 31.

```
(triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   "http://www.w3.org/2002/07/owl#equivalentProperty")
   (object    "http://www.w3.org/2002/07/owl#SymmetricProperty")
)
```

Figure 30. Declare property equivalence as a symmetric property

```
(defrule equivalent-property (declare (salience 100))
   (triple
     (predicate "http://www.w3.org/2002/07/owl#equivalentProperty")
     (subject   ?p1)
     (object    ?p2))
   (triple (predicate p1?) (subject ?s) (object ?o))
   (not (triple (predicate p2?) (subject ?s) (object ?o)))
   =>
   (assert (triple (predicate p2?) (subject ?s) (object ?o)))
)
```

Figure 31. Rule for forward-chaining completion of property equivalence

As far as the OWL meta-model is concerned, we are focusing on those aspects of OWL-Lite relevant to our application scenarios. More precisely the current system handles: `Resource`, `Class`, `Property`, `type`, `subClassOf`, `subPropertyOf`, `ObjectProperty`, `TransitiveProperty`, `SymmetricProperty`, `inverseProperty`, `equivalentProperty`, `equivalentClass`, `sameIndividualAs`, `DatatypeProperty`, `FunctionalProperty`, `InverseFunctionalProperty`. – the source code and the results obtained by running our OWL engine on the official OWL test cases are available at http://mycampus.sadehlab.cs.cmu.edu/public_pages/OWLEngine.html.

Likewise, triples in the ontologies and annotations loaded into the e-Wallet are asserted as unordered facts. Finally, domain-dependent rules are also loaded in the e-wallet. An example of one such rule is illustrated in Figure 32. It defines colleagues as members of the same team. Such rules can help represent and interpret context-sensitive preferences such as ‘My colleagues can see my location when I am at work’.

```
                                                                                OWL
<rowl:Rule direction="forward">
  <rdfs:label>Members of the same group means colleagues</rdfs:label>
  <rowl:head>
    <mc:Person rdf:about="&variable;#person1">
      <mc:colleague rdf:resource="&variable;#person2"/>
    </mc:Person>
  </rowl:head>
  <rowl:body>
    <mc:Team rdf:ID="&variable;#group">
      <mc:include><mc:Person rdf:about="&variable;#person1"/></mc:include>
      <mc:include><mc:Person rdf:about="&variable;#person2"/></mc:include>
    </mc:Team>
  </rowl:body>
</rowl:Rule>
                                                                   XSLT
-----------------------------------------------------------------------------------
                                                                             CLIPS
(defrule Members_of_the_same_group_means_colleagues
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?group)
   (object    "http://mycampus.cs.cmu.edu/ontology#Team")
  )
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#include")
   (subject   ?group)
   (object    ?person1)
  )
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person1)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#include")
   (subject   ?group)
   (object    ?person2)
  )
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person2)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
   =>
 (assert
  (triple
   (predicate "http://www.w3.org/1999/02/22-rdf-syntax-ns#type")
   (subject   ?person1)
   (object    "http://mycampus.cs.cmu.edu/ontology#Person")
  )
 )
 (assert
  (triple
   (predicate "http://mycampus.cs.cmu.edu/ontology#colleague")
   (subject   ?person1)
   (object    ?person2)
  )
 )
)
```

Figure 32. Rule defining colleagues as members of the same team

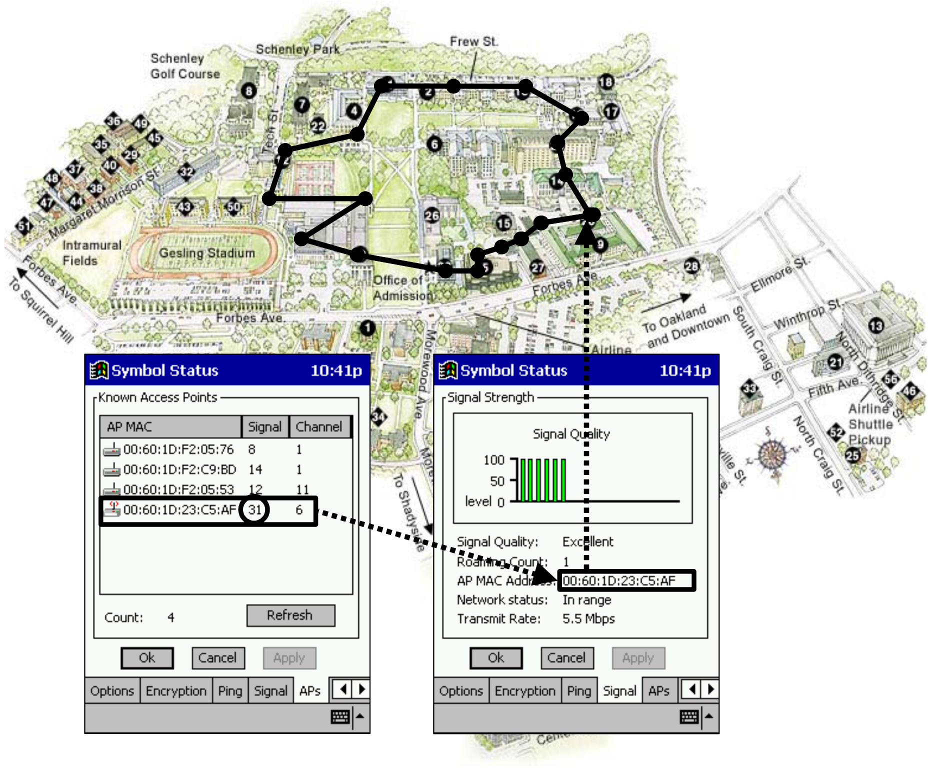
The inference engine is used to complete the base applying all the rules, thus saving time during the query solving process and providing a rollback point if needed.

9.8 Service Layer Processing

As indicated earlier, needs for service triples can be satisfied by either migrating a matching core triple or by activating a matching service invocation rule. For obvious efficiency reasons, it makes sense to always look for core triples first. This can be enforced by assigning a high priority (also called salience) to rules that look for matching core triples and lower priority to service invocation rules. Service invocation rules are themselves given different priorities, based on the nature of the resource they invoke. This is further detailed below:

- If the needed service triple can be obtained by invoking an internal function (e.g. getting the current time), that function will be activated;
- If no internal service can provide the triple (or if internal calls have failed) but there is a personal service that can possibly provide the needed triple (e.g., obtaining the user's current activity from her personal calendar), the corresponding backward invocation rule is fired, calling that personal resource's Web Service wrapper;
- If no personal resource can provide the needed triple or if the calls failed, the engine looks for invocation rules involving public web services. This can include invoking public Semantic Web search engines (e.g. CORESE [CZC02] or distributed search architecture such as TAP [GMM03]) or public matchmaking services such as the one in [PKPS02] – this step is not currently implemented.
- If everything fails, the query is considered to have failed.

In summary, the body of each rule requires a need for a particular piece of information or triple (e.g. Fabien's location) along with the availability of a specific set of arguments (e.g. knowledge of the IP address of Fabien's PDA). When these conditions are matched, the rule fires and calls the service (Figure 33 depicts the semantic web service used to support location-tracking over CMU's wireless LAN).



Known Access Points

AP MAC	Signal	Channel
00:60:1D:F2:05:76	8	1
00:60:1D:F2:C9:BD	14	1
00:60:1D:F2:05:53	12	11
00:60:1D:23:C5:AF	31	6

Count: 4

Signal Strength

Signal Quality:	Excellent
Roaming Count:	1
AP MAC Address:	00:60:1D:23:C5:AF
Network status:	In range
Transmit Rate:	5.5 Mbps

Annotation of access points on campus:

```
<mc:AccessPoint>
  <mc:MAC>00:60:1D:23:C5:AF</mc:MAC>
  <mc:location rdf:resource="http://www.cmu.edu/SmithHall" />
</mc:Entity>
```

Result of invoking the location-tracking web service:

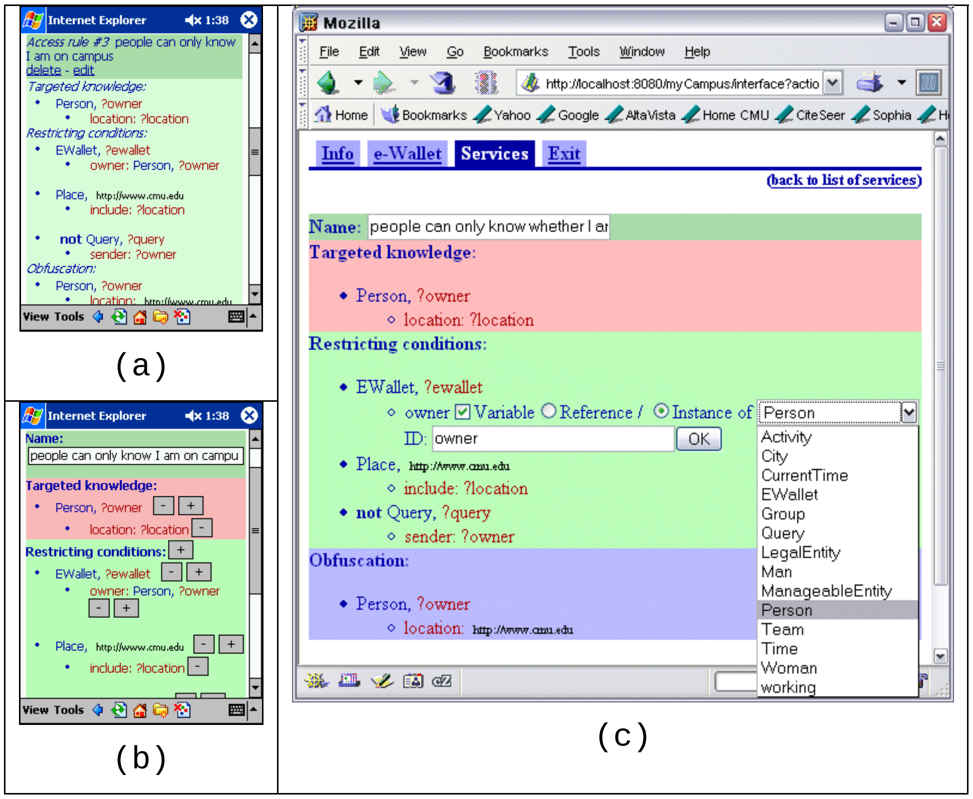
```
<mc:Entity rdf:ID="http://cs.cmu.edu/~fgandon">
  <mc:location rdf:resource="http://www.cmu.edu/SmithHall" />
</mc:Entity>
```

Figure 33. Semantic web service for location-tracking over CMU's wireless LAN.

When looking for particular piece of information, rule salience helps determine the order in which to try and invoke available services (e.g. if multiple sources of location information are available). In general, we envision having a set of rules, where, should everything else fail, the e-Wallet reverts to a low salience rule that invokes one or more semantic search engines and/or one or more public matchmaking services. Clearly, as users acquire new personal resources (e.g. a new calendar), they will have to register them with their e-Wallets (e.g. using predefined service profiles that are provided with the resource itself).

9.9 Capturing User Preferences

As should be clear by now, our Semantic Web technologies are capable of capturing a wide variety of user preferences that may refer to any relevant set of OWL ontologies. This is true for message filtering preferences, food preferences, music preferences, privacy preferences, scheduling preferences, *etc.* One approach to capturing these preferences is to develop a variety of special-purpose editing tools that enable users to specify their preferences with regard to predefined sets of ontologies. For instance, each time a user subscribes to (or acquires) a new task-specific agent, she might be prompted by a special-purpose editor to customize a predefined set of preferences. The same could be done to capture predefined sets of privacy preferences. However, a key objective in our architecture has been to provide for an open environment, where new sources of contextual information, new contextual ontologies and new agents can be introduced over time. Supporting the capture of user privacy preferences in this broader context ideally requires a general-purpose privacy preference editor that enables users to refer to any relevant source of contextual information and any relevant contextual ontology. Figure 34 shows screenshots of such a general-purpose privacy preference editor. The editor uses XSLT stylesheets and allows users to browse (Figure 34-a) and edit their privacy rules (Figure 34 - b and c).



Access rule #3 people can only know I am on campus
delete - edit
Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- EWallet, ?ewallet
  - owner: Person, ?owner
- Place, http://www.cmu.edu
  - include: ?location
- **not** Query, ?query
  - sender: ?owner

Obfuscation:
- Person, ?owner
  - location: http://www.cmu.edu

Name: people can only know I am on campu
Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- EWallet, ?ewallet
  - owner: Person, ?owner
- Place, http://www.cmu.edu
  - include: ?location

View Tools
(c)
Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/myCampus/interface?actio
Home Bookmarks Yahoo Google AltaVista Home CMU CiteSeer Sophia
Info e-Wallet Services Exit
(back to list of services)

Name: people can only know whether I ar
Targeted knowledge:
- Person, ?owner
  - location: ?location

Restricting conditions:
- EWallet, ?ewallet
  - owner [✓ Variable] ○ Reference / [● Instance of] Person
    ID: owner OK
- Place, http://www.cmu.edu
  - include: ?location
- **not** Query, ?query
  - sender: ?owner

Obfuscation:
- Person, ?owner
  - location: http://www.cmu.edu

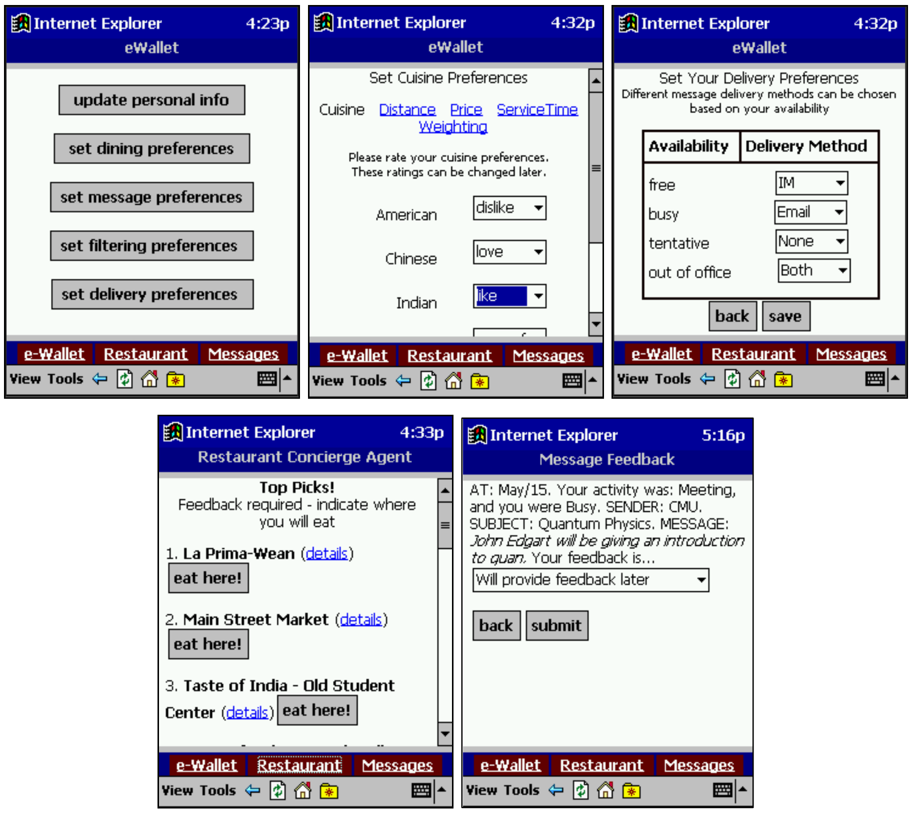
Figure 34. Generic rule editor that enables users to (a) browse and (b) (c) edit their OWL-based privacy/confidentiality preferences.

The editor allows users to create new rules as well as edit and delete existing ones. The editor draws directly on available ontologies (ontologies loaded into the e-Wallet), enabling users to express any privacy/confidentiality rules they want as they relate to concepts and properties defined in these ontologies. The editor takes into account the OWL meta-model as the user edits rules. For instance, it will restrict the instantiation of a given concept to be within the range of a given property, as specified using the OWL "ObjectProperty" construct [OWL].

Every single editing operation is specified through an external XSLT stylesheet. The stylesheets are independent of the domain ontologies and could be refined to support more specific instantiations of our rule editor. In addition, rule editing is supported through the definition of high-level functions, namely "creating", "deleting", "extracting", "updating" a rule, or "adding/deleting concepts", "adding/deleting properties", *etc*. These high-level functions are instantiated at run time, using XSLT stylesheets that perform the actual manipulation. In other words, the editor could easily be adapted to accommodate extensions to our rule syntax. As can also be seen, use of this general purpose privacy preference editor, in its current form, is best left to system administrators and advanced users.

9.10 Empirical Evaluation

An early version of our architecture has been validated in *my*Campus, a context-aware environment aimed at enhancing everyday campus life at CMU. The environment is accessible to members of the campus community from their PDAs over the university's wireless LAN. An example of a *my*Campus agent we have developed is a "restaurant concierge" that gives users suggestions on where to have lunch, depending on their food preferences, their location on campus and the weather. For instance, when it rains, the concierge might look for places that do not require walking outside – depending on how the user sets her preferences. Another task-specific agent that has proved particularly popular among students is a context-aware message filtering agent. The agent filters incoming alerts, taking into account a profile of topics a user is interested in as well as contextual attributes such as the user's current activities (e.g. "When in class, only show me emergency alerts" or "When I am busy, delay showing me interesting messages until my current activity is over"). Screenshots of both agents are shown in Figure 35.


Internet Explorer 4:23p
eWallet
update personal info
set dining preferences
set message preferences
set filtering preferences
set delivery preferences
e-Wallet Restaurant Messages
View Tools

Internet Explorer 4:32p
eWallet
Set Cuisine Preferences
Cuisine Distance Price ServiceTime Weighting
Please rate your cuisine preferences.
These ratings can be changed later.

American	dislike
Chinese	love
Indian	like

e-Wallet Restaurant Messages
View Tools

Internet Explorer 4:32p
eWallet
Set Your Delivery Preferences
Different message delivery methods can be chosen based on your availability

Availability	Delivery Method
free	IM
busy	Email
tentative	None
out of office	Both

back save
e-Wallet Restaurant Messages
View Tools

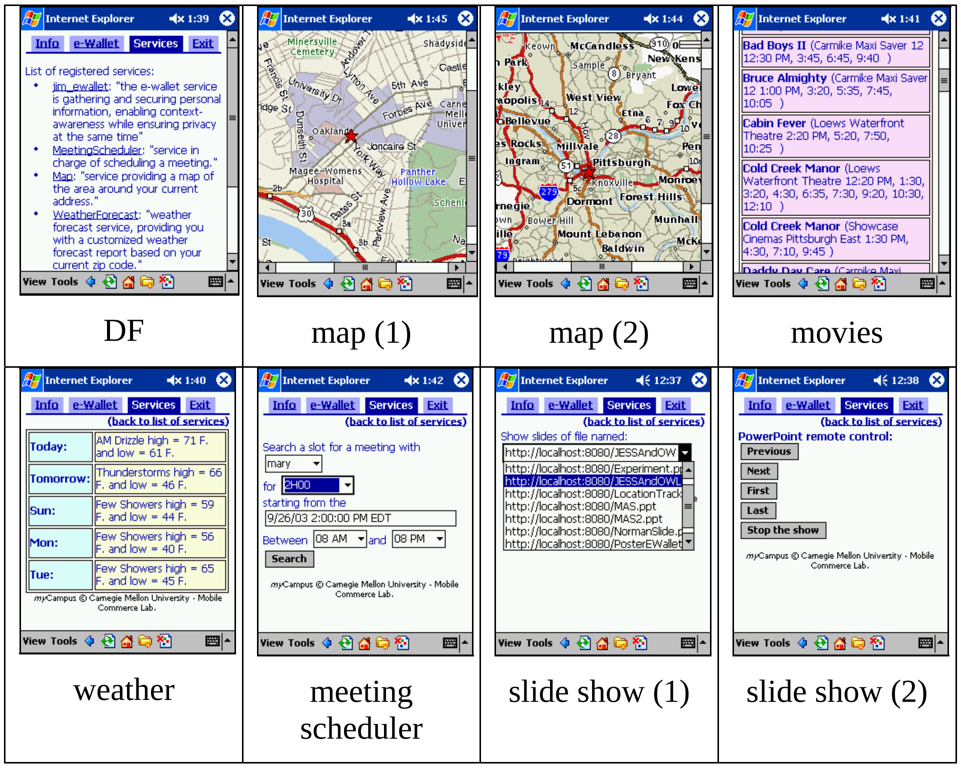
Internet Explorer 4:33p
Restaurant Concierge Agent
Top Picks!
Feedback required - indicate where you will eat
1. La Prima-Wean (details) eat here!
2. Main Street Market (details) eat here!
3. Taste of India - Old Student Center (details) eat here!
e-Wallet Restaurant Messages
View Tools

Internet Explorer 5:16p
Message Feedback
AT: May/15. Your activity was: Meeting, and you were Busy. SENDER: CMU. SUBJECT: Quantum Physics. MESSAGE: *John Edgart will be giving an introduction to quan.* Your feedback is...
Will provide feedback later
back submit
e-Wallet Restaurant Messages
View Tools

Figure 35. Screenshots of the e-Wallet (top), restaurant recommendation from the Restaurant Concierge Agent (bottom left) and request for feedback from the Message Filtering Agent (bottom right)

Evaluation of the system by 11 users over a period of 3 days has indicated positive overall user acceptance. Among other things, the experiments required users to configure their individual preferences and use several context-aware agents, including a context-aware restaurant concierge agent and a context-aware message filtering agent. The message filtering agent was used to process a total of 44 messages for each user and the restaurant concierge was systematically used by students to decide where to eat, selecting from a total of 23 web services created for restaurants on or near campus. The context-aware functionality embedded in the agents used in the experiments proved rather successful with context awareness systematically improving performance over the use of static user profiles. For instance, detailed feedback from users indicated that over 70 percent of the 484 messages processed by the filtering agents benefited from the use of contextual information. In other words, the action taken by the message filtering agent based on contextual information was always at least as good as that taken based on static user profiles and was actually better in 70 percent of the cases.

Figure 17 displays additional screenshots of the *my*Campus environment, illustrating different sets of pervasive computing scenarios. The Directory Facilitator (DF) provides users with a list of available task-specific agents. The map agent is a user locator that displays the location of a user on a map, subject to that user's privacy preferences. Map (1) corresponds to a request where the user is willing to disclose the particular zipcode she is in, while map (2) corresponds to a query where she is only willing to disclose her location at the level of the city she is in. Other similar agents include a location-sensitive movie recommendation agent and a location-sensitive weather forecast agent. A slide show agent enables users to access slides that other users have agreed to share with them subject to preferences specified in their e-Wallets and to display these slides on a nearby projector.



List of registered services:
- jim_ewallet: "the e-wallet service is gathering and securing personal information, enabling context-awareness while ensuring privacy at the same time"
- MeetingScheduler: "service in charge of scheduling a meeting."
- Map: "service providing a map of the area around your current address."
- WeatherForecast: "weather forecast service, providing you with a customized weather forecast report based on your current zip code."

Bad Boys II (Carmike Maxi Saver 12 12:30 PM, 3:45, 6:45, 9:40)

Bruce Almighty (Carmike Maxi Saver 12 1:00 PM, 3:20, 5:35, 7:45, 10:05)

Cabin Fever (Loews Waterfront Theatre 2:20 PM, 5:20, 7:50, 10:25)

Cold Creek Manor (Loews Waterfront Theatre 12:20 PM, 1:30, 3:20, 4:30, 6:35, 7:30, 9:20, 10:30, 12:10)

Cold Creek Manor (Showcase Cinemas Pittsburgh East 1:30 PM, 4:30, 7:10, 9:45)

Daddy Day Care (Carmike Maxi

(back to list of services)

Today:	AM Drizzle high = 71 F. and low = 61 F.
Tomorrow:	Thunderstorms high = 66 F. and low = 46 F.
Sun:	Few Showers high = 59 F. and low = 44 F.
Mon:	Few Showers high = 56 F. and low = 40 F.
Tue:	Few Showers high = 65 F. and low = 45 F.

*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

(back to list of services)

Search a slot for a meeting with mary for 2H00 starting from the 9/26/03 2:00:00 PM EDT Between 08 AM and 08 PM

Search

*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

(back to list of services)

Show slides of file named:

http://localhost:8080/JESSAndOW
http://localhost:8080/Experiment.p
http://localhost:8080/JESSAndOWL
http://localhost:8080/LocationTrack
http://localhost:8080/MAS.ppt
http://localhost:8080/MAS2.ppt
http://localhost:8080/NormanSlide.p
http://localhost:8080/PosterEWallet

(back to list of services)

PowerPoint remote control:

Previous
Next
First
Last
Stop the show

*my*Campus © Carnegie Mellon University - Mobile Commerce Lab.

Figure 36. *my*Campus: Additional screenshots showcasing additional agents and pervasive computing scenarios

.

9.11 Summary and Concluding Remarks

In this paper, we have presented a Semantic Web architecture for context-awareness and privacy. A key element of our architecture is its e-Wallet, which supports the automated discovery and access of a user’s personal resources subject to user-specified privacy preferences. Personal and public resources are represented as web services. Service invocation rules along with service ontologies and service profiles enable the e-Wallet to dynamically identify the most promising resources available to answer a query. When one resource is unavailable, service invocation rules can help identify the next most relevant resource (e.g. using a calendar resource instead of a location tracking resource to estimate the user’s current location). Public matchmaking services and semantic search engine functionality can also be leveraged through low salience rules that amount to reverting to these services when everything else has failed. Another innovation introduced in our e-Wallet is its support for a rich set of privacy preferences, including obfuscation rules that enable users to selectively adjust the accuracy or inaccuracy of responses they provide depending on the context of each query. We have described a three-layer implementation of our e-Wallet using JESS, OWL-Lite and XSLT stylesheets. A query to the e-Wallet successively results in the creation of needs for authorization triples and service triples. The latter can be satisfied through the identification of matching core triples and/or the activation of service invocation rules.

Experiments with *my*Campus indicate that different students are interested in different task-specific agents and that, to be effective, many agents require access to a great variety of contextual resources. Our experiments also confirmed that users are concerned about protecting access to their personal information. The need for leveraging a variety of contextual attributes and the students’ demand for privacy strongly argue for Semantic e-Wallets such as the one presented here. A key challenge however remains to reconcile the power of these Semantic environments with all important usability requirements that demand systems which are flexible, yet easy to configure. We are experimenting with different approaches to editing and learning user profiles, which we hope will help alleviate this problem.

10 Appendix B: A More In-Depth Overview of Meta-Control Strategies to Interleave Semantic Web Reasoning and Service Discovery

This appendix is based on our article on "Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies" [SR05a].

Abstract

In many domains, users and organizations need to protect their information and services subject to policies that reflect dynamic, context-sensitive considerations. More generally, enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policy elements (e.g. finding an appropriate source of location information to enforce a location-sensitive access control policy). In this paper, we introduce a semantic web framework for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more Policy Enforcing Agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning and service discovery and access. This research has been conducted in the context of *my*Campus, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University.

10.1 Introduction

The increasing reliance of individuals and organizations on the Web to help mediate a variety of activities is giving rise to a demand for richer security and privacy policies and more flexible mechanisms to enforce these policies. Enterprises want to selectively expose core business functionality and sensitive business information to various partners based on the evolving nature of their relationships (e.g. disclosing rough product specifications to prospective suppliers versus disclosing more detailed requirements to actual suppliers, or giving selective visibility into the company's inventory positions or demand forecasts to preferred supply chain partners). Employees in a company may be willing or required to share information about their whereabouts or about their activities with some of their team members or their boss but only under some conditions (e.g. during regular business hours or while on company premises). Coalition forces may need to selectively share sensitive intelligence information but only to the extent it is relevant to a specific joint mission. Each of these examples illustrates the need for what we generically refer to as context-sensitive security and privacy policies, namely policies whose conditions are not tied to static considerations but rather conditions whose satisfaction, given the very same actors (or principals), will likely fluctuate over time. Enforcing such policies in open environments is particularly challenging for several reasons:

- Sources of information available to enforce these policies may vary from one principal to another (e.g. different users may have different sources of location tracking information made available through different cell phone operators)
- Available sources of information for the same principal may vary over time (e.g. when a user is on company premises her location may be obtained from the wireless LAN location tracking functionality operated by her company as well as through her cell phone operator, but when she is not on company premises the cell phone operator is the only option – subject to relevant privacy policies she may have specified)
- Available sources of information may not be known ahead of time (e.g. new location tracking functionality may be installed or the user might roam into a new area)

Accordingly, enforcing context-sensitive policies in these open domains requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection and access of relevant sources of contextual information. This requirement exceeds the capability of decentralized management infrastructures proposed so far and calls for privacy and security enforcing mechanisms capable of operating according to significantly less scripted scenarios than is the case today (e.g. [BSF02,HSSK04,LGC+05]). It also calls for much richer service profiles than those found in early web service standards.

In this paper, we introduce a semantic web framework for dynamically interleaving policy reasoning and external service identification, selection and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more Policy Enforcing Agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to opportunistically interleave semantic web reasoning and service discovery and access. In this paper, we focus on a particular type of Policy Enforcing Agent we refer to as Information Disclosure Agents. These agents are responsible for enforcing two types of policies: access control policies and obfuscation policies. The latter are policies that manipulate the accuracy or inaccuracy with which information is released (e.g. disclosing whether someone is busy or not rather than disclosing what they are actually doing). The research reported herein has been conducted in the context of MyCampus, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University [SCV+03,GS03,GS04a].

The work presented in this paper builds on concepts of decentralized trust management developed over the past decade [BFL96]. Most recently, a number of researchers have started to explore opportunities for leveraging the openness and expressive power associated with Semantic Web and agent frameworks in support of decentralized trust management (e.g. [UPC+03, KFJ03, KPS04, HKL+04, APM04, UBJ04, DKF+05] to name just a few). Our own work in this area has involved the development of Semantic e-Wallets that enforce context-sensitive privacy and security policies in response to requests from context-aware applications implemented as intelligent agents [GS03, GS04a]. In this paper, we introduce a significantly more decentralized framework, where policies can be distributed among any number of agents and web services. Within this framework, we present a meta-control architecture for interleaving semantic web reasoning and web service discovery in enforcing context-sensitive privacy and security policies.

The remainder of this paper is organized as follows. Section 2 introduces an overall architecture for distributing and enforcing privacy and security policies, using a pervasive computing context to illustrate how these policies can be deployed in practice. It follows with an overview of our Information Disclosure Agent, detailing its different modules and how their operations are opportunistically orchestrated in response to incoming requests. A motivating example based on the pervasive computing environment introduced earlier is presented in Section 10.2. Section 10.3 details our query status model, which serves as a basis to our meta-control strategies. Section 10.4 describes our service discovery model. Some implementation issues are discussed in Section 10.5. Concluding remarks are provided in Section 10.6.

10.2 Overall Approach and Architecture

10.2.1 Pervasive Computing as an Application Context

To help put things in perspective, we consider a pervasive computing environment, where each user interacts with the infrastructure either directly (e.g. walking into a room, entering the subway system) or indirectly via agents to which they delegate tasks (e.g. a general-purpose user-agent such as a micro-browser on a PDA or cell phone, policy evaluation and notification agents, or task-specific agents such as a context-aware message filtering agent or a meeting scheduler agent). The infrastructure provides a set of resources generally tied to different geographical areas, such as printers, surveillance cameras, campus-based location tracking functionality, and so on (see Figure 37). These resources are all modeled as services that can be automatically discovered based on rich ontology-based service profiles advertised in service directories and accessed via open APIs. In general, services can offer functionality and/or serve as sources of contextual information. A camera service, a calendar system, or a location tracking service are examples that can offer both. Services can also build on one another, with simple services providing building blocks for the definition of more complex ones. An example is the “printer service” in Figure 37, which itself relies on the “find nearest printer service,” which in turn relies on a “people locator service” to find the location of the user. The “people locator service” in turn might be able to dynamically select from a number of possible services available to locate people such as a badge system or a combination of a system of cameras along with a video analysis service. Each service and agent has an owner, whether an individual or an organization, who is responsible for setting policies for the service or agent.

Services that collect information about users may broadcast disclosure messages that inform target users (or more specifically their agents) about the operation of the service (e.g. users who enter a smart room or the subway system). Some disclosures are one-way announcements: they simply inform the user that information is collected about them and possibly how that information is used.

Other disclosure messages may give the user some options. For example, a location-tracking service may give the user the choice of opting out. Alternatively, the user may be able to allow tracking, while limiting the use of his or her location information (e.g. only for emergency use) or she may require that all requests for her location be cleared with her own Information Disclosure Agent (enforcing her regular privacy and security policies). A Policy Disclosure Evaluation Agent may respond to disclosures automatically, based on the user's policies (e.g. opting out). The same agent may also be able to occasionally notify its user of policies that might lead her to modify her behavior, as well as prompt its user to manually select among possible options when needed.

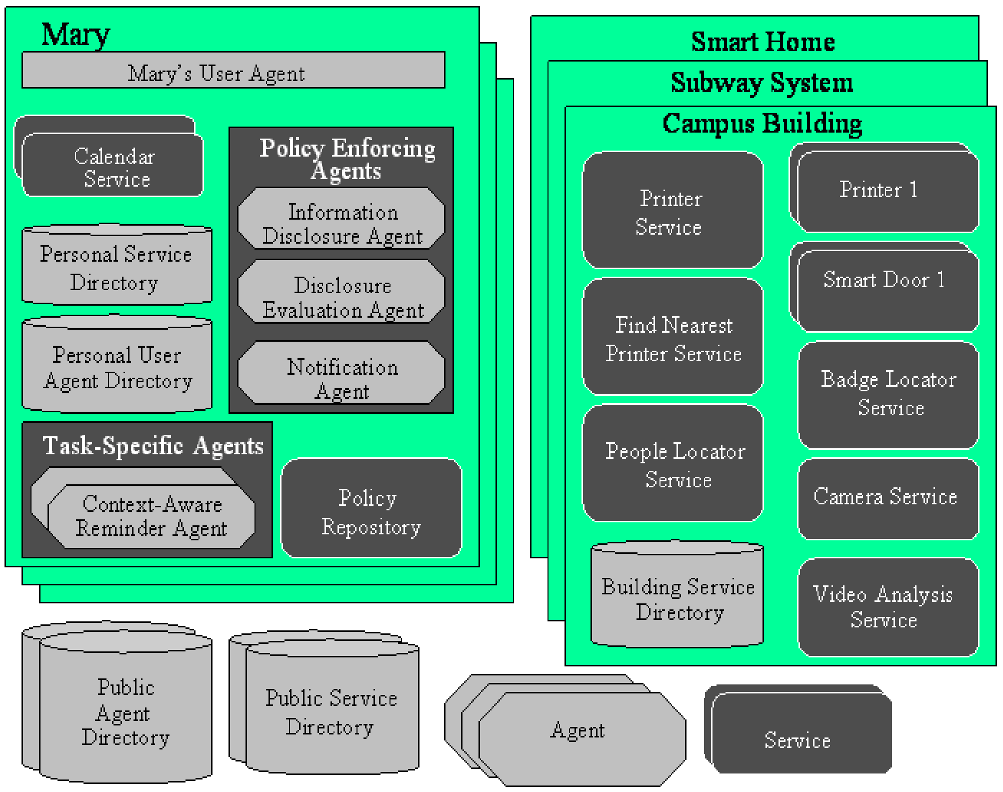


Figure 37. Pervasive Computing as an Application Domain

Each entity (or principal) in the system (whether an individual, a service, an agent or an organization) has a set of credentials and a set of policies. These policies can include:

- Access control policies that limit access only to entities that can be proved to satisfy certain conditions.
- Obfuscation policies that associate different levels of accuracy or inaccuracy to different sets of credentials.
- Information collection policies (a la P3P [CLM+02], that specify what type of information is collected by a service, for what purpose, how that information will be stored, etc.
- Notification Preference Policies specifying under which conditions a user may want to be alerted about the presence of sensors or other information collection applications.

Collectively, these policies enable users and organizations to manage their privacy practices, specifying what information they are willing to disclose (access control) and at what level of granularity (obfuscation) and notifying users or their agents about the information they collect and what happens to that information. Policy enforcement is delegated to different sets of agents (these agents may occasionally request input or feedback from their users, as already illustrated earlier). For the sake of clarity, in the remainder of this paper, we focus more specifically on one such type of agent, namely an *Information Disclosure Agent* responsible for enforcing both access control and obfuscation policies. The architecture presented for this agent can however be adapted to implement a number of other context-sensitive Policy Enforcing Agents such as the ones illustrated in Figure 37.

10.2.2 Information Disclosure Agent: An Example of a Policy Enforcing Agent

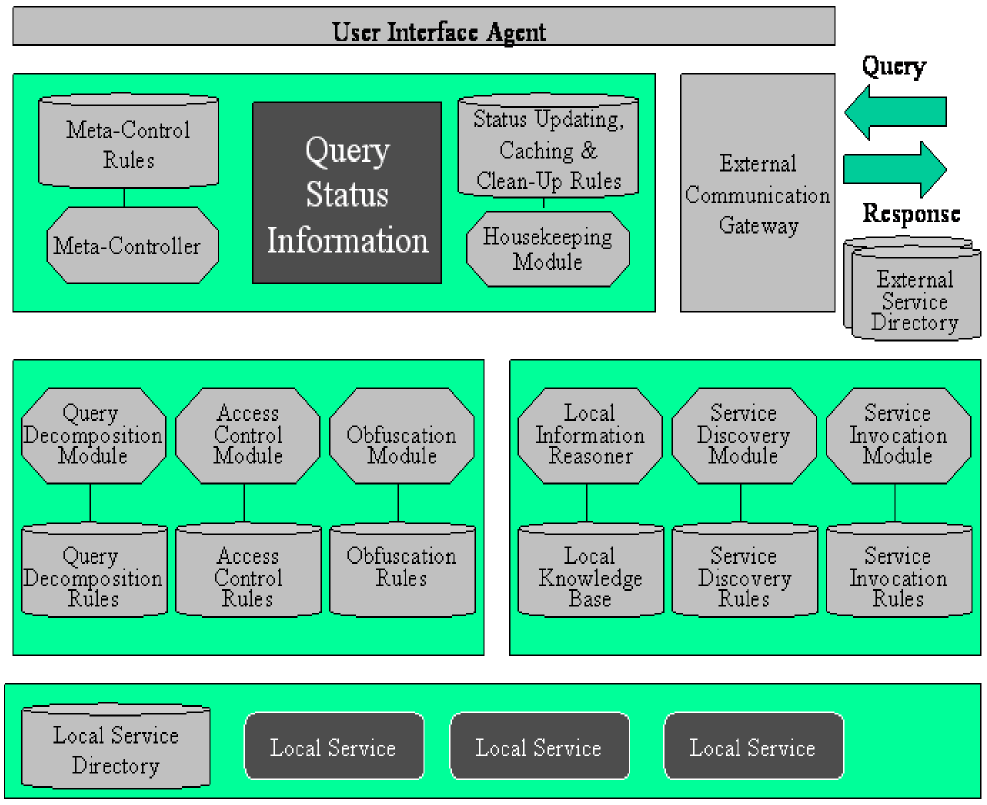


Figure 38. Information Disclosure Agent: Logical Architecture

An Information Disclosure Agent (IDA) processes incoming requests (e.g. a query about the location of the agent's owner or a request to access a service under the owner's control) subject to a set of access control and obfuscation polices captured in the form of rules. As it processes incoming queries, the agent records status information that helps it monitor its own progress in enforcing its policies and in obtaining the necessary information. Based on this updated query status information, a meta-control module ("meta-controller") dynamically orchestrates the operations of modules it has at its disposal to process queries (Figure 38). As these modules report on the status of activities they have been tasked to perform, this information is processed by a Housekeeping module responsible for updating query status information (e.g. changing the status of a query from being processed to having been processed). Simply put, the agent continuously cycles through the following three basic steps:

1. The meta-controller analyzes the latest query status information and invokes one or more modules to perform particular tasks. As it invokes these modules the meta-controller also updates relevant query status information (e.g. update the status of a

query from “not yet processed” to “being processed”). All query status information includes timestamps.

2. Modules complete their tasks (whether successfully or not) and report back to the Housekeeping module – occasionally modules may also report on their ongoing progress in handling a task
3. The Housekeeping module updates detailed status information based on information received from modules and performs additional housekeeping activities (e.g. caching the results of recent requests to mitigate the effects of possible denial of service attacks, cleaning up status information that has become irrelevant, etc.)

For obvious efficiency reasons, while an IDA consists of a number of logical modules, each operating according to a particular set of rules, it is actually implemented as a single reasoning engine. In our current work we use JESS [Fri03], a high-performance Java-based rule engine that supports both forward and backward chaining – the latter by reifying "needs for facts" as facts themselves, which in turn trigger forward-chaining rules. The following provides a brief description of each of the modules orchestrated by the IDA’s meta-controller – note that other types of Policy Enforcing Agents typically entail different sets of modules:

- *Query Decomposition Module:* This module takes as input a particular query and breaks it down into elementary needs for information, which can each be thought of as subgoals or sub-queries. We refer to these as *Query Elements*.
- *Access Control Module* is responsible for determining whether a particular query or sub-query is consistent with relevant access control policies – modeled as access control rules. While some policies can be checked just based on facts contained in the agent’s local knowledge base, many policies require obtaining information from a combination of both local and external services. When this is the case, rather than immediately deciding whether or not to grant access to a query, the Access Control Module requests additional facts – *also Query Elements*. These requests are added to the agent’s Query Status Information Knowledge Base along with information about their parent Query or Query Element – namely the Query or Query Element for which they are needed.
- *Obfuscation Module* sanitizes information requested in a query according to relevant obfuscation policies – also modeled as rules. As it evaluates relevant obfuscation policies, this module too can post request for additional information (Query Elements) to the Query Status Information Knowledge Base (via the Housekeeping Module).
- *Local Information Reasoner*: This reasoner corresponds to “static” domain knowledge (facts and rules) known locally to the IDA or at least knowledge that does not change too frequently (e.g. the name and email address of the agent’s owner, possibly a list of friends and family members, etc.)
- *Service Discovery Module*: This module helps the IDA identify promising sources of information to complement its local knowledge. This includes both *local services* and *external services*. Local services can be identified through a local service directory (e.g. a directory of services under the direct control of the agent’s owner such as a calendar service running on his desktop or on his smart phone). External services can be identified through external service directories (whether public or not). Communication with external service directories takes place via the agent’s

External Communication Gateway. Rather than relying solely on searching service directories, the service discovery module also allows for the specification of what we refer to as *service identification rules*. These rules directly map information needs on prespecified services (whether local or external). An example of such rule might be: "when looking for current activity, try first my calendar service". When available, such rules can yield significant performance improvements, while allowing the module to revert to more general service directory searches when they fail. We assume that all service directories rely on OWL-S to advertise service profiles (see Section 6).

- *Service Invocation Module:* This module allows the agent to invoke relevant services, whether local or external. It is important to note that, in our architecture, each service can have its own Information Disclosure Agent (IDA). As requests are sent to services, their IDAs may in turn respond with requests for additional information to enforce their own policies.
- *User Interface Agent:* The meta-controller treats its user as just another module who is modeled both as a potential source of domain knowledge (e.g. to acquire relevant contextual information) as well as a potential source of meta-control knowledge (e.g. if a particular query takes too long to process, the user may be requested whether it is worth expending additional computational resources processing that query or not).

Modules support one or more services that can each be invoked by the meta-controller along with relevant parameter values. For instance, the meta-controller may invoke the query decomposition module and request it to decompose a particular query; it may invoke the access control module and task it to proceed in evaluating access control policies relevant to a particular query; etc. In addition, meta-control strategies do not have to be sequential. For instance, it may be advantageous to implement *meta-control strategies* that enable the IDA to concurrently request the same or different facts from several services. These and other issues are further discussed in Section 7.

10.3 Sample Scenario

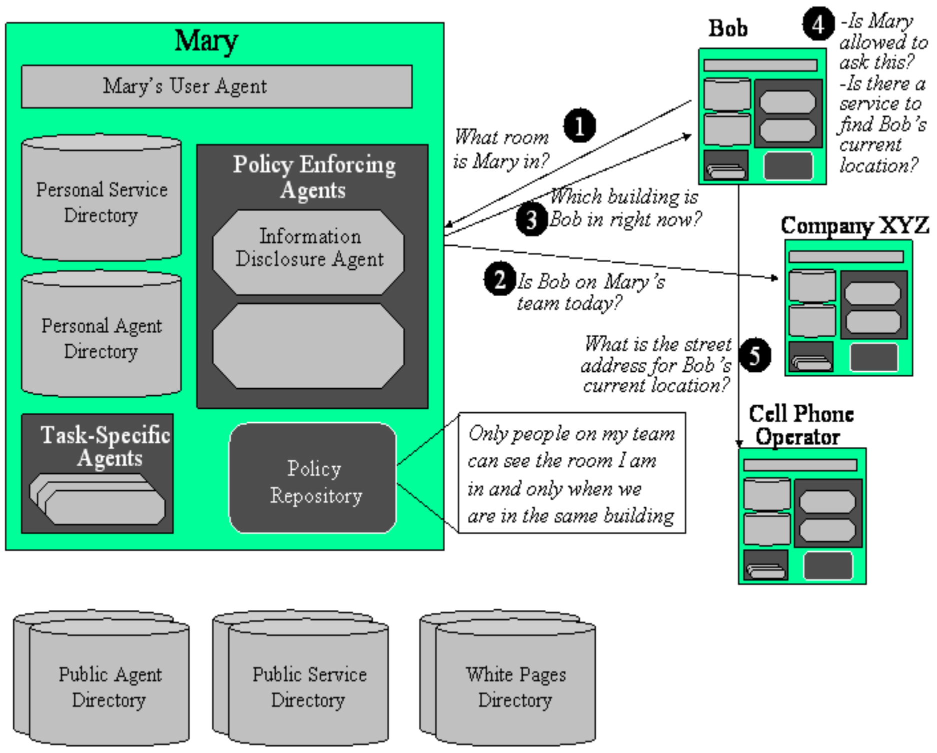

Mary
Mary's User Agent
Personal Service Directory
Personal Agent Directory
Policy Enforcing Agents
Information Disclosure Agent
Task-Specific Agents
Policy Repository
Only people on my team can see the room I am in and only when we are in the same building
What room is Mary in?
1
Which building is Bob in right now?
3
2 Is Bob on Mary's team today?
What is the street address for Bob's current location?
5
Bob
4 -Is Mary allowed to ask this? -Is there a service to find Bob's current location?
Company XYZ
Cell Phone Operator
Public Agent Directory
Public Service Directory
White Pages Directory

Figure 39. Illustration of first few steps involved in processing a request from Bob to find out about the room Mary is in.

The following scenario will help illustrate how IDAs operate. Consider Mary and Bob, two colleagues who work for company XYZ. Mary and Bob are both field technicians who constantly visit other companies. Mary's team changes from one day to the next depending on the nature of her assignment. Mary relies on an Information Disclosure Agent to enforce her access control policies. In particular, she has specified that she is only willing to disclose the room that she is in to members of her team and only when they are in the same building. Suppose that today Bob and Mary are on the same team and that Bob is querying Mary's IDA to find out about her location. For the purpose of this scenario, we assume that Mary and Bob are visiting Company ABC and are both in the same building at the time the query is issued. Both Bob and Mary have cell operators who can provide their location at the level of the building they are in – but not at a finer level. Upon entering Company ABC, Mary also registered with the company's location tracking service, which operates over the wireless LAN and is compatible with her WiFi-enabled smart phone. As she registered with the service, one of her Policy Enforcing Agents (her Policy Disclosure Evaluation Agent) negotiated that all requests about her location be redirected to her IDA. For the purpose of this scenario, we also assume that Mary's IDA does not yet know whether Bob is on her team. It therefore needs to identify a service that can help it determine whether this is the case. A service discovery step helps identify a service operated by Company XYZ (Bob and Mary's employer) that contains up-to-date information about teams of field technicians. This step requires a directory with rich semantic service profiles, describing what each service does (e.g. type of information it can provide, level of accuracy or recency, etc.). To be interpretable by agents such as Mary's IDAs, these profiles also need to refer to concepts specified in shared ontologies (e.g. concepts such as projects, teams, days of the week, etc.). Once Mary's IDA has determined that Bob is on her team today, it proceeds to determine whether they are in the same building by asking Bob's IDA about the building he is in. Here Bob's IDA goes through a service discovery step of its own and determines that a location tracking service offered by his cell phone operator is adequate. Completion of the scenario involves a few additional steps of the same type. Note that in this scenario we have assumed that Mary's IDA trusts the location information returned by Bob's IDA. It is easy to imagine scenarios where her IDA would be better off looking for a completely independent source of information. It is also easy to see that these types of scenarios can also lead to deadlocks. In later sections, we briefly discuss elements of our architecture that partially helps mitigate these problems (e.g. query status update information that keeps track of the origin of requests for information – see Section 5 below).

10.4 Query Status Model

The IDA's *Meta Controller* relies on meta-control rules to analyze query status information and determine which module(s) to activate next. Meta-control rules are currently modeled in CLIPS. In other words, each meta-control rule is an if-then clause with a LHS (left hand side) specifying its premises and a RHS (right hand side) its conclusions. More specifically, LHS elements of meta-control rules refer to query status information, while RHS ones contain facts that result in module activations. While both LHS and RHS are expressed in CLIPS they refer to queries received by the IDA and to *query elements* generated while processing these queries. A query element is a need for elementary information required to fully process a query (e.g. finding someone's location or calendar activity to help answer a more complex query). Queries themselves are expressed in an extension of OWL (see [GS04a]). Query status information in the LHS relies on a taxonomy of predicates that helps the agent keep track of queries and query elements - e.g., whether a query has been or is being processed, what individual query elements it has given rise to, whether these elements have been cleared by relevant access control policies and sanitized according to relevant obfuscation control policies. Query status information helps keep track of how far along the IDA is in obtaining the information required by each query element, whether the agent's local knowledge base has been consulted, whether local or external services have been identified and consulted, etc. It also enables the agent to keep track of dependencies between queries and query elements. This information can help identify potential deadlocks. All query status information is time stamped, enabling the meta-controller to also implement rules that take into account how much time has already been spent trying to process a query, clearing access control policies or waiting for an external service to respond. A sample of query status information predicates is provided in Table 1. This list is just illustrative and will be used to revisit the scenario introduced in Section 4. Clearly, different taxonomies of predicates can lead to more or less sophisticated meta-control strategies. For the sake of clarity, status predicates in Table 1 are organized in six categories: 1) communication;

2) query; 3) query elements; 4) access control; 5) obfuscation and 6) information collection. Status information is represented in CLIPS with status predicates and a number of slots detailing particular pieces of status information. Typical slots include:

– **A query ID** or **query element ID** to which the predicate refers
– **A parent query ID** or **parent query element ID** to help keep track of dependencies (e.g. a query element may be needed to help check whether another query element is consistent with a context-sensitive access control policy). These dependencies, if passed between IDA agents, can also help detect deadlocks (e.g. two IDA agents each waiting for information from the other to enforce their policies)
– **A time stamp** that describes when the status information was generated or updated. This information is critical when it comes to determining how much time has elapsed since a particular module or external service was invoked. It can help the agent look for alternative external services or decide when to prompt the user (e.g. to decide whether to wait any longer).

	Sample Status Predicates	**Description**
1)	Query-Received	Query received. A related queries slot helps determine the query's context and identify potential deadlocks.
	Sending-Response	Response to a query is being sent
	Response-Sent	Response has been successfully sent
	Response-Failed	Response failed (e.g. message bounced back)
2)	Processing Query	Query is being processed
	Query Decomposed	Query has been decomposed (into primitive query elements)
	All-Elements-Available	All query elements are available (i.e. the information they require is available)
	All-Elements-Cleared	All query elements have been cleared by relevant access control policies
	Clearance-Failed	Failed to clear one or more access control policies
	All-Elements-Sanitized	All query elements have been sanitized according to relevant obfuscation policies
	Sanitization-Failed	Failed to pass one or more obfuscation policies
	Element-Needed	A query element is needed. Query elements may result from the decomposition of a query or may be needed to enforce policies. The query element's origin helps distinguish between these different cases
	Processing-Element	A need for a query element is being processed
	Element-Available	Query element is available

3)	Element-Cleared	Query element has been cleared by relevant access control policies
	Clearance-Failed	Failed to pass one or more access control policies
	Element-Sanitized	Query element has been sanitized according to relevant obfuscation policies
	Sanitization-Failed	Failed to pass one or more obfuscation policies
4)	Clearance-Needed	A query or query element needs to be cleared by relevant access control rules
5)	Sanitization-Needed	Query or query element has to be sanitized subject to relevant obfuscation policies
6)	Check-Condition	Check whether a condition is satisfied. Special type of query element.
	Element-not-locally-available	The value of a query element can not be obtained from the local knowledge base
	Element-need-service	A query element requires the identification of a relevant service
	No-service-for-Element	No service could be identified to help answer a query element. This predicate can be refined to differentiate between different types of services (e.g. local versus external)
	Service-identified	One or more relevant services have been identified to help answer a query element
	Waiting-for-service-response	A query element is waiting for a response to a query sent to a service (e.g. query sent to a location tracking service to help answer a query element corresponding to a user's location)
	Failed-service-response	A service failed to provide a response. Again this predicate could be refined to distinguish between different types of failure (e.g. service down, access denied, etc.)
	service-response-available	A response has been returned by the service. This will typically result in the creation of an "Element-Available" status update.

Table 1. Sample list of status information predicates.

Query status information updates are asserted as new facts (with old information being cleaned up by the IDA's housekeeping module – Figure 38). As query updates come in, they trigger one or more meta-control rules, which in turn result in additional query status information updates and the eventual activation of one or more of the IDA's modules. An example of a simple meta-control rule to activate the service discovery module if information about the room that Mary could not be obtained locally (from the local information reasoner) can be expressed as follows:

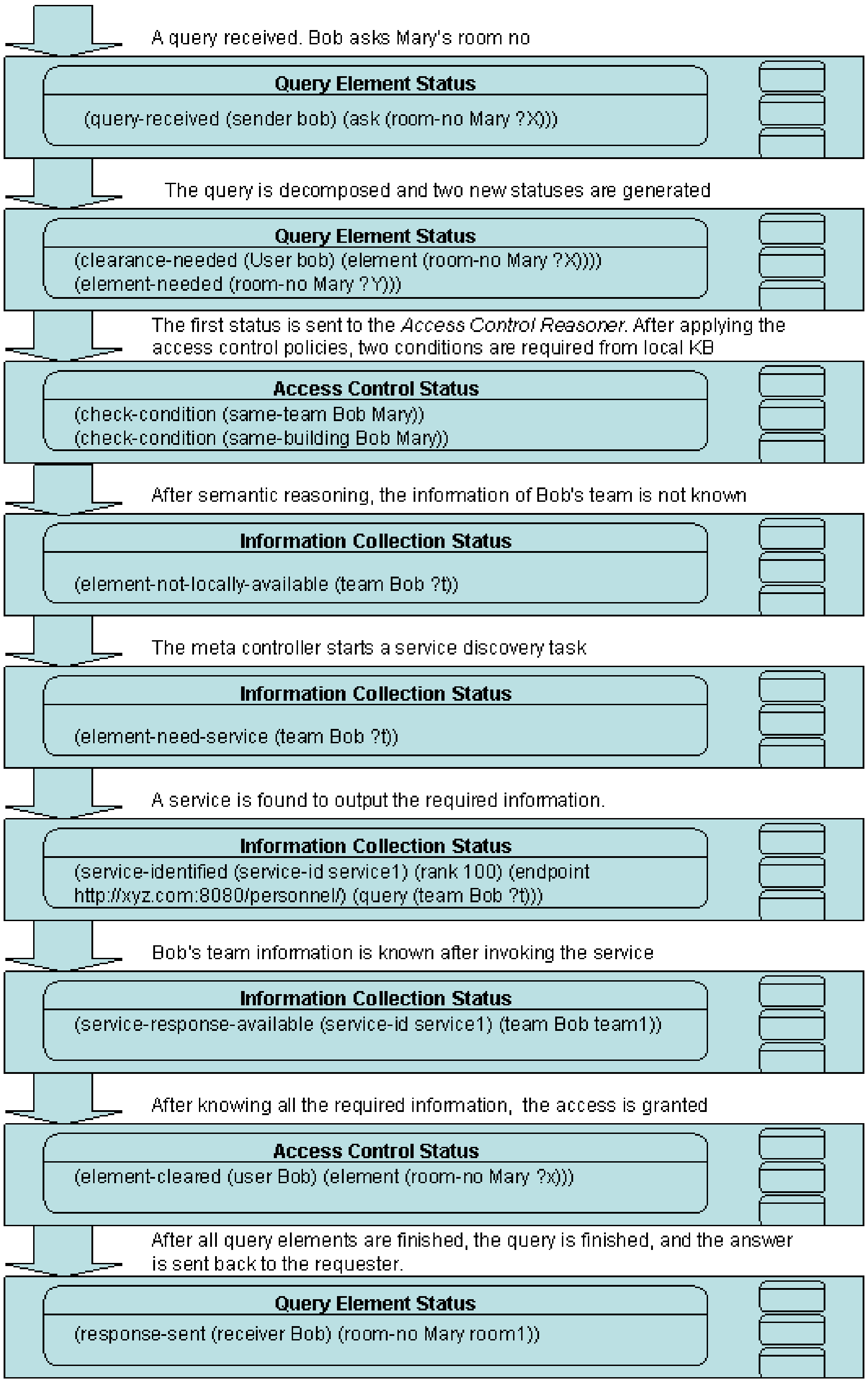

A query received. Bob asks Mary's room no
Query Element Status
(query-received (sender bob) (ask (room-no Mary ?X)))
The query is decomposed and two new statuses are generated
Query Element Status
(clearance-needed (User bob) (element (room-no Mary ?X))))
(element-needed (room-no Mary ?Y)))
The first status is sent to the *Access Control Reasoner*. After applying the access control policies, two conditions are required from local KB
Access Control Status
(check-condition (same-team Bob Mary))
(check-condition (same-building Bob Mary))
After semantic reasoning, the information of Bob's team is not known
Information Collection Status
(element-not-locally-available (team Bob ?t))
The meta controller starts a service discovery task
Information Collection Status
(element-need-service (team Bob ?t))
A service is found to output the required information.
Information Collection Status
(service-identified (service-id service1) (rank 100) (endpoint http://xyz.com:8080/personnel/) (query (team Bob ?t)))
Bob's team information is known after invoking the service
Information Collection Status
(service-response-available (service-id service1) (team Bob team1))
After knowing all the required information, the access is granted
Access Control Status
(element-cleared (user Bob) (element (room-no Mary ?x)))
After all query elements are finished, the query is finished, and the answer is sent back to the requester.
Query Element Status
(response-sent (receiver Bob) (room-no Mary room1))

Figure 40. An example of status changes

(element-needed (parent-id ?x) (elem-id ?y) (room Mary ?z))

(element-not-locally-available (elem-id ?y) (room Mary ?z1))

=>

(assert (module service-discovery) (element-need-service (elem-id ?y) (output (room Mary ?z))))

In practice, meta-control rules are typically more general than this (i.e. they don't just refer to the room Mary is in).

Example

The following illustrates the processing of a query by an IDA, using the scenario introduced in Figure 39. Figure 40 depicts some of the main steps involved in processing a request from Bob about the room Mary is in, highlighting some of the main query status information updates. Specifically, Bob's query about the room Mary is in isprocessed by the IDA's *Communication Gateway*, resulting in a query information status update indicating that a new query has been received:

(query-received (queryid 1) (sender Bob) (ask (room-no Mary ?X)))

The meta-controller proceeds by invoking the *Query Decomposition Module*, resulting in the creation of two query elements – for the sake of simplicity we omit Mary's obfuscation policy: one to establish whether this request is compatible with Mary's access control polcies and the other to obtain the room she is in:

(clearance-needed (parent-id 1) (elem-id 1.1) (User Bob) (element (room-no Mary ?x)))

(element-needed (parent-id 1) (elem-id 1.2) (room-no Mary ?X))

The meta-controller decides to first focus on the "clearance-needed" query element and invokes the *Access Control Module*. This module determines that two conditions need to be checked and accordingly creates two new query elements ("check-conditions"):

(check-condition (parent-id 1.1) (elem-id 1.1.1) (same-team Bob Mary))

(check-condition (parent-id 1.1) (elem-id 1.1.1) (same-building Mary Bob))

The first condition requires checking whether Bob and Mary are on the same team, while the second one is to determine whether Bob is in the same building as Mary. Each condition requires a series of information collection steps that are orchestrated by the meta-control rules in Mary's IDA. In this example, we assume that the IDA's local KB contains a semantic reasoning rule:

```
(team ?p1 ?t)
(team ?p2 ?t)
=>
(same-team ?p1 ?p2)
```

We also assume that the IDA knows Mary's team but not Bob's. According the following query status information update is generated:

```
(element-not-locally-available (parent-id 1.1.1) (elem-id 1.1.1.1) (team Bob ?t))
```

Mary's IDA has a meta-control rule to initiate service discovery when a query element can not be found locally. The rule is of the form:

```
(element-needed (elem-id ?x) ?y)
(element-not-locally-available (elem-id ?x) ?y)
=>
(assert (module discover) (element-need-service (parent-id ?x) (elem-id ?z) ?y))
```

Thanks to this rule, the *Service Discovery Module* is now activated. A service to find Bob's team is identified (e.g. a service operated by company XYZ). This results in a Query Status Information update of the type "service-identified". If there are multiple matching services, they may be ranked and the top service is invoked (multiple services could also be invoked concurrently).

```
(service-identified (elem-id ?e) (service-id ?s1) (rank ?r1) (endpoint ?e1) ?x)
(not (service-identified (elem-id ?e) (service-id ?s2) (rank ?r2) (endpoint ?e2) ?x))
(leq ?r1 ?r2)
=>
(assert (module invocation) (invoke-service (parent-id ?e) (elem-id ?ee) (service-id
?s1) (endpoint ?r1) ?x))
```

We assume that the service returns the team that Bob is in. The Housekeeping module updates the necessary Query Status Information, indicating among other things that information about Bob's team has been found ("element-available") and cleaning old status information. This is done using a rule of the type:

```
?n <-(element-needed (elem-id ?e) ?y)

(service-response-available (parent-id ?e) (elem-id ?ee) (service-id ?s) ?a)

=>

(retract ?n)

(assert (module meta) (element-available (parent-id ?ee) (elem-id ?eee) ?a))
```

The scenario continues through a number of similar steps.

10.5 The Service Discovery Model

A central element of our architecture is the ability of IDA agents to dynamically discover sources of information (whether local or external) to help obtain the information needed by Query Elements. Sources of information are modeled as Semantic Web Services and may operate subject to their own access control and obfuscation policies enforced by their own IDA agents. Accordingly service invocation is itself implemented in the form of queries sent to a service's IDA agent. .

Service Model

Each service (or source of information) is described by a *ServiceProfile* in OWL-S [OWLS]. *ServiceProfiles* consist of three parts: (1) information about the provider of the service, (2) information about the service's functionality and (3) information about non-functional attributes [SEH02]. Functional attributes include the service's inputs, outputs, preconditions and effects. Non-functional attributes are other properties such as accuracy, quality of service, price, location, etc. An example of a location tracking service operated on the premises of Company ABC is described in Figure 41.

```
<profileHierarchy:InformationService
                              rdf:ID="PositioningService ">
   <!-- reference to the service specification -->
  <service:presentedBy
          rdf:resource="&Service;#PositioningService"/>
   <profile:has_process rdf:resource="&Process;#PositionProc"/>
  <profile:serviceName>Positioning_Service_in_ABC
  </profile:serviceName>
```

```
    <!-- specification of quality rating for profile -->
    <profile:qualityRating>
      <profile:QualityRating rdf:ID="SERVQUAL">
              <profile:ratingName>SERVQUAL</profile:ratingName>
              <profile:rating rdf:resource="&servqual;#Good"/>
      </profile:QualityRating>
    </profile:qualityRating>
          <profile:hasPrecondition
                rdf:resource="&Process;#LocInABC"/>
   <profile:hasOutput
               rdf:resource="&Process;#RoomNoOutput"/>
</profileHierarchy:InformationService>
```

Figure 41. An example service profile in OWL-S

Because in our architecture service invocation is done by submitting queries to a service's IDA, our service profiles currently do not include inputs. Instead, services obtain their input parameters by submitting queries back to the requester. In practice, this process can become somewhat inefficient and we plan to also investigate more sophisticated discovery models that examine required service input requirements in light of the IDA's access control and obfuscation policies.

Service outputs are represented as OWL classes, which play the role of typing mechanism for concepts and resources. Using OWL also allows for some measure of semantic inference as part of the service discovery process. If an agent requires a service that produces a contextual attribute of a specific type as output, then all services that output the value of that attribute as a subtype are potential matches.

Service preconditions and effects are also used for service matching. For instance., the positioning service in Figure 41 has a precondition specifying that it is only available on company ABC's premises.

10.6 Implementation

Our policy enforcing agents are based on JESS, a high-performance rule-based engine implemented in Java. Domain knowledge, including service profiles, queries, access control policies and obfuscation policies are expressed in either in OWL or in extensions of OWL [GS04a]. XSLT transformations are used to translate OWL facts and extensions of OWL (to model rules and queries) into CLIPS (Figure 15). Query status information and meta-control rules are directly expressed in CLIPS. Agent modules are organized as JESS modules. Rules in a JESS module only fire when that module has the focus and only one module can be in focus at a time. Currently all information exchange between agents is done in the clear and without digital signatures. In the future, we plan to use SSL or some equivalent protocol for all information exchange..

10.7 Concluding Remarks

In many domains, users and organizations need to protect their information and services subject to policies that reflect dynamic, context-sensitive considerations. More generally, enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policy elements. In this paper, we presented a semantic web framework for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more *Policy Enforcing Agents* responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning and service discovery and access.

The Information Disclosure Agent presented in this paper is just one instantiation of our more general concept of Policy Enforcing Agents. Other policies (e.g. information collection policies, notification preference policies) will typically rely on slightly different sets of modules and different meta-control strategies, yet they could all be implemented using the same type of architecture and many of the same principles presented in this paper. Our Policy Enforcing Agents rely on a taxonomy of query information status predicates to monitor their own progress in processing incoming queries and enforcing relevant security and privacy policies. They use meta-control rules to decide which action to take next (e.g. decomposing queries, seeking local or external information, etc.). This work is conducted in the context of *my*Campus, a context-aware environment aimed at enhancing everyday campus life at Carnegie Mellon University [SCV+03,GS04a]. Experiments with an early implementation of our framework seem promising. At the same time, it is easy to see that the generality of our framework also gives rise to a number of challenging issues. Future work will focus on testing the scalability of our framework, evaluating tradeoffs between the expressiveness of privacy and security policies we allow and associated computational and communication requirements. Other issues of particular interest include studying opportunities for concurrency (e.g. simultaneously accessing multiple web services), dealing with real-time meta-control issues (e.g. deciding when to give up or when to look for additional sources of information/web services), and breaking deadlocks [LNOS04].

11 Appendix C: Using Semantic Web Services for Planning-Enabled Context-Aware Mobile Applications

This appendix is based on our article on "Using Semantic Web Services for Context-Aware Mobile Applications" [SSG04]. It includes a discussion of AI planning functionality that takes advantage of semantic web service discovery, access and composition.

Abstract

One way of overcoming the challenges associated with mobile and pervasive computing environments involves providing users with higher levels of automation. This in turn requires capturing the context within which the user operates. In this paper, we describe ongoing research aimed leveraging Semantic Web Services in support of context awareness. This includes modeling sources of contextual information as web services that can be automatically discovered and accessed by agents that assist the user with different sets of tasks. By automatically discovering and accessing a variety of external web services, these agents can automatically develop and execute simple plans to assist the user (e.g. ordering a pizza, organizing an evening out, etc.). This research is being conducted in the context of *my*Campus, a prototype semantic web environment to enhance everyday campus life at Carnegie Mellon University.

11.1 Introduction

With hundreds of Internet-enabled mobile devices, the mobile Internet is opening the door to a slew of new mobile applications and services that will assist users as they engage in time-critical, goal-driven tasks [Sad02]. Yet today, the mobile commerce landscape is dominated by relatively simple infotainment services. Moving beyond these simple services requires overcoming the inherent input/output limitations of mobile devices through higher degrees of automation and the development of services that understand the *context* within which their users operate – e.g. their locations, the activities they are engaged in, who their friends and colleagues are as well as a number of other contextual attributes and preferences. In this paper we look at ways of using contextual information using web services and automatically chaining together multiple services to achieve complex tasks using planning. Our usage of web services is discussed in the context of *my*Campus-a context-aware environment aimed at enhancing everyday campus life.

11.2 Overview of MyCampus

In *my*Campus, users can acquire (or subscribe) to different sets of task-specific agents that help them with different tasks. To properly operate these agents require knowledge of one or more contextual attributes about their users as well as possibly other users. These attributes can potentially be acquired from a number of possible resources, which typically vary from one user to another (e.g. not everyone uses Microsoft Outlook as their calendar) and may even vary over time for the same user. To overcome this challenge, sources of contextual information in *my*Campus are modeled as Semantic Web Services that can automatically be discovered and accessed by agents. Access to a user's contextual resource is controlled according to user-specified privacy preferences (including context-sensitive preferences such as colleagues have access to my location only if they have a meeting with me in the next hour).

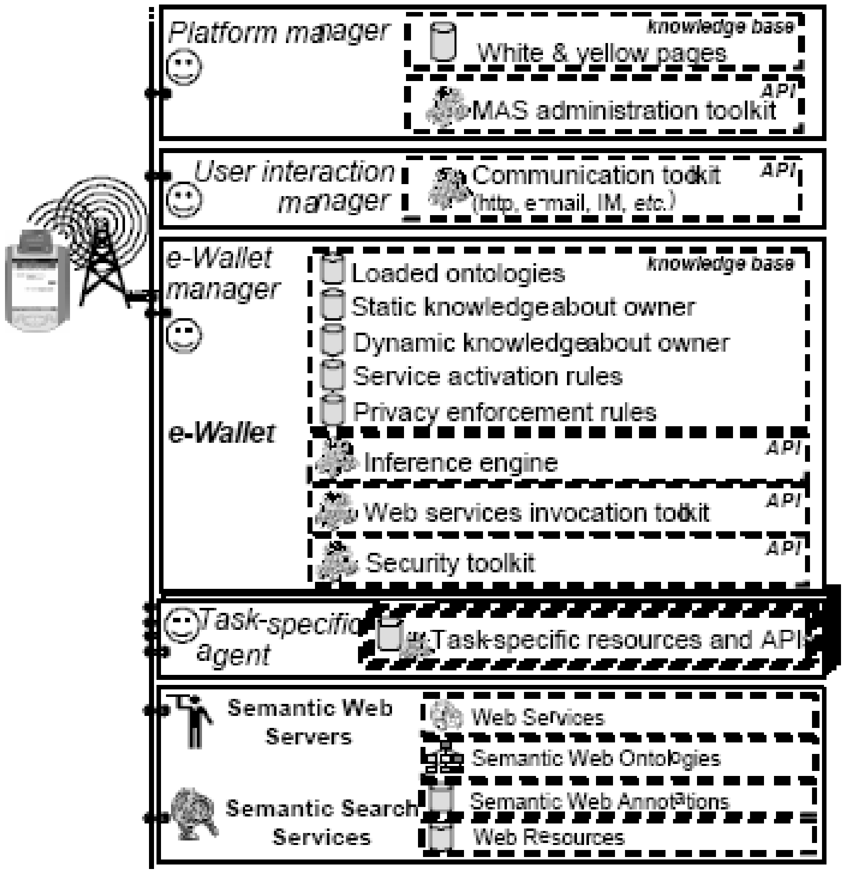


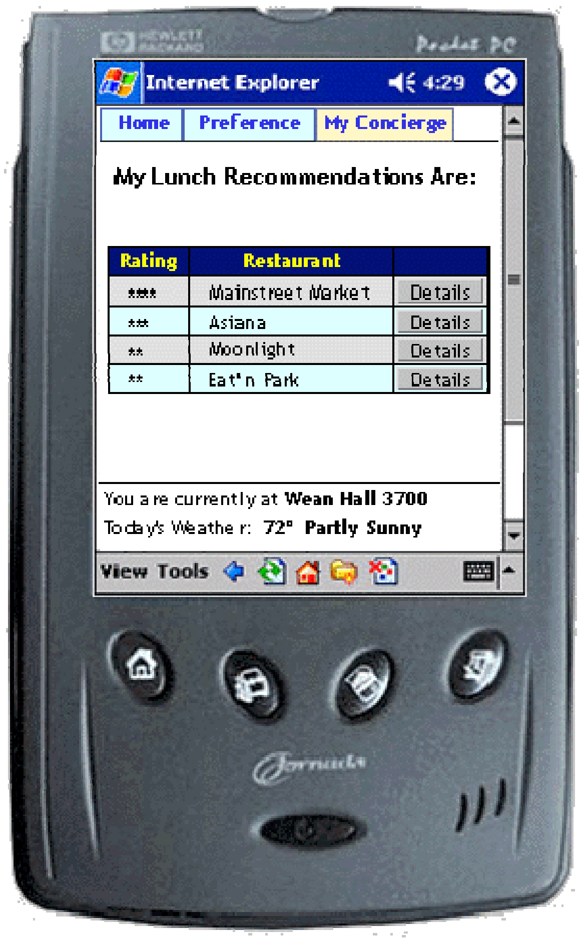
Figure 42. myCampus architecture: a user's perspective – the smiley faces represent agents.

The e-Wallet Manager (or simply e-Wallet) serves as a repository of static knowledge about the user – just like.NET Passport, except that here knowledge is represented using OWL [OWL]. In addition, the e-Wallet contains knowledge about how to access more information about the user by invoking a variety of resources, each represented as a Web Service. This knowledge is stored in the form of rules that map different contextual attributes onto one or more possible service invocations, enabling the e-Wallet to automatically identify and activate the most relevant resources in response to queries about the user's context (e.g. accessing the user's calendar to find out about her availability, or consulting one or more location tracking applications in an attempt to find out about her current location). User-specified privacy rules, also stored in the e-Wallet, ensure that information about the user is only disclosed to authorized parties, taking into account the context of the query. They further adjust the accuracy of the information provided in accordance with the user's obfuscation preferences rules. (For example, I am willing to disclose my location in the building to Bob but I only want to reveal the city information to Mary).

Figure 42 provides an overview of *my*Campus. It illustrates a situation where access is from a PDA over a wireless network. However, our architecture extends to fixed Internet scenarios and more generally to environments where users can connect to the infrastructure through a number of access channels and devices – information about the particular access device and channel can actually be treated as part of the user's context and be made available through her e-Wallet.

Clearly, agents are not limited to accessing information about users in the environment. Instead, they also typically access public Web Services, Semantic Web annotations, public ontologies and other public resources. On CMU's campus, where we have deployed *my*Campus, this includes access to a variety of services such as 23 restaurant web services or a pubic weather forecasting web service.

In the following sections, we focus on the use of web services in our system. Additional details on *my*Campus and some of the agents we have deployed can be found in [GS03, GS04a, SCV+03].



My Lunch Recommendations Are:

Rating	Restaurant	
****	Mainstreet Market	Details
***	Asiana	Details
**	Moonlight	Details
**	Eat'n Park	Details

You are currently at **Wean Hall 3700**

Today's Weather: 72° **Partly Sunny**

Figure 43. This Restaurant Concierge is an example of a myCampus agent.

11.3 Using Web Services for Contextual Information

This section discusses in detail, how web services are used as sources of contextual information. As explained in an earlier section, contextual attributes can potentially be acquired from a number of possible resources, which typically vary from one user to another and may even vary over time for the same user. In *my*Campus, sources of contextual information are wrapped as Semantic Web Services [DAMLS]. This means that each source of contextual information is described by a profile that describes its functional properties in relation to one or more ontologies [Gru93]. For instance, Microsoft Outlook Calendar is an instance of a resource that provides both calendar activity information and user location information. Service descriptions also include details about how to invoke a service (e.g. input, output and preconditions). Thanks to these profiles, relevant sources of contextual information can be automatically discovered and accessed.

11.3.1 Service Invocation Rules

One particularly efficient way of identifying a service that can provide information about a given contextual attribute (e.g. user's location) is by using a set of rules**.** Service invocation rules provide a mapping between contextual attributes and personal resources available to access these attributes, viewing each personal resource as a Semantic Web Service. For instance, the location of the user is provided from a service wrapped around the user's Microsoft Outlook calendar. Service invocation rules are not limited to providing a one-to-one mapping between contextual attributes and personal resources. Instead, they can leverage rich ontologies of personal resources, enabling the e-Wallet to select among a number of possible personal resources based on availability, accuracy and other relevant considerations. For instance, in response to a query about the user's location, the rules can specify that, when the user is driving, the best method available is the GPS in her car. If she is at work and her wireless-enabled PDA is on, her location can be obtained using location tracking functionality running over the enterprise's wireless LAN. If everything else fails, her calendar might have some information about her location.

11.3.2 Semantic Web Services Using OWL-S

Binding contextual attributes to services through rules is a very efficient mechanism, especially for time critical applications. However, this is not always practical, especially considering that, over time, users may acquire new task-specific agents and new sources of contextual information. Maintaining detailed service invocation rules that cover all possible situations is simply impractical. Instead, a more flexible (albeit more computationally demanding) approach involves relying on automated service discovery. In general, contextual information about a given user is obtained by sending a query to her eWallet. As detailed in the following subsections, the eWallet then relies on a combination of local service identification rules and local and global service discovery mechanisms to identify one or more relevant sources of contextual information. In the process, it also ensures that the request is compatible with the user's privacy rules, as detailed in [FN71]. Service discovery is done as follows. Services are registered in directories along with profiles that describe their various relevant capabilities and characteristics. Unfortunately, current Web Services standards for discovery such as UDDI [UDDI] are not sufficient. UDDI does not describe a service in terms of the functionalities it offers but provides information about the entity that owns it and provides mechanisms to classify the service in terms of standard taxonomies such as North American Industry Classification System (NAICS) [NAICS]. Moreover it supports only syntactic matching. A directory that advertises the functional attributes of the service is a far better alternative. Moreover if these descriptions are represented in Semantic Web Languages then subsumption-based reasoning can be used for semantic matching of service functionalities. The OWL ontology for services (OWL-S) [OWLS] framework can be used to build directories that contain service descriptions (functional attributes expressed in Semantic Web Languages). More information on semantic discovery of services can be found in [LH03]. Information in the eWallet is stored as semantic annotations, therefore; the eWallet can readily make use of OWL-S (see appendix) based semantic service discovery. To further clarify the need for semantic discovery let us use the example of a service that provides a list of Italian restaurants using zip code as input. The user of the eWallet could be looking for an *food:eatery* using a popular ontology called *food*. Let us assume that OWL-S is used to represent the service as an atomic service with *loc:zipcode* as the input and *food:ItalianRestaurant* as the output. Also, assume that the *food* ontology categorizes *ItalianRestaurant* as a *subclassOf* of *eatery*. By using semantic inferencing one can conclude that all instances of *ItalianRestaurant* are instances of *eatery* and therefore the service in question is returned as a match to the query. This kind of matching is not possible in UDDI because firstly, inputs and outputs are not specified and secondly, the UDDI framework is not capable of semantic reasoning which we used to make match in the above example.

11.4 Service Composition

Composition is defined as the task of putting together atomic/basic services to perform complex tasks. To start the discussion on composition, let us first consider an example. When a user wants to find about the local weather forecast, the system needs to first find the current location of the user before it invokes the weather service. The location of the user is in turn obtained by the eWallet of the user which actually invokes a web service to obtain this information. This in itself is a primitive form of composition where the weather service requires the invocation of the location service. This composition was possible using input and output **"type"** matching. One could think of a sequence of two invocations in which the second service can be invoked only when the invocation of the first operator produce a particular **"value"**. Such cases cannot be handled by type matching. To capture this ordering constraint we make use of preconditions. Preconditions are interpreted as: when a precondition is associated with a service, the service can be invoked only when the precondition is satisfied. A precondition is satisfied based on the value of output with which it is associated. We use both, type and value matching for composition.

Services are described using inputs, outputs and preconditions (IOPs). We model services in our system using the OWL-S *process* ontology. More specifically, all services are described using OWL-S *atomic processes*. We also propose the addition of a new construct - *realizedBy* which connects preconditions to outputs. Whether the output actually realizes the precondition can be found out only during service invocation. We use this construct to first compose and invoke the services involved later. This technique of disconnecting composition and invocation enables us to find out whether the existing set of services can satisfy the user' goal before we invoke a single service. If one were to implement simultaneous composition and invocation, the search for operators would have to be in the forward direction i.e., from the initial state to the goal. The main problem with *forward-chaining* arises due to high branching factor which leads to a huge search space. This also leads to other complications. One has to invoke services to move towards the goal. In the end, if the goal cannot be achieved, the service invocations achieved nothing but the waste of computation and network bandwidth. The situation is potentially worse if some of these services are *non-idempotent*. Thus our approach of first using *backward-chaining* to build the plan and then invoking the services is efficient and avoids the complications caused by *non-idempotent* services.

11.4.1 Automatic Operator Extraction and Planning

The composition of atomic services into complex services can be viewed as planning. This is done by mapping atomic services into planning operators and running a planning algorithm to link these operators. The plan generated using these operators constitutes a complex service. As mentioned before, *my*Campus environment has various specific agents. Some of these perform simple tasks like providing weather information to the user whereas others can do more complex tasks like ordering food. Some of these complex agents make use of the planner. The user specifies the goal using an interface designed for each task-specific agent. On receiving the goal, the task-specific agent checks whether the information resides in the eWallet of the user or if it can be provided by services known to the agent. If both these checks fail, the discovery phase is initiated. If an appropriate service(s) is found, the task of converting atomic services into planning operators is done as follows. A service description consists of one or more atomic processes. The service descriptions are first transformed into Predicate-Subject-Object (PSO). For each atomic service, we build *STRIPS*-style [FN71] planning operators by querying the triples. The actual composition is done by a simple *backward-chaining* planning algorithm.

The composition problem can be transformed into a planning problem given by the tuple ***<Op, G>*** where ***Op*** is the set of operators obtained from service descriptions and ***G*** is the user's goal(s). Composition is done using the following algorithm:

```
Compose (Goals G, Operators Op)
   0. If G is empty
     a. Return Success
   1. Search Op for operators
       that satisfy all the goals G.
   2. IF no operator is found
     a. Return Fail
   3. ELSE
     a. Delete G
     b. Add unavailable inputs of operator to G'
     c. Add unavailable outputs (using realizedBy)
     d. Compose(G', Op)
```

The first call to the Compose routine consists of all available operators and the user's goal. The algorithm looks for services that satisfy the goal. If no such service is found, composition fails and discovery can be re-initiated to look for more operators. If on the other hand, all goals in G are satisfied by operator(s), then the system checks to find if inputs to these operators are available. All unavailable inputs are transformed into goals for the next iteration. A similar availability check is run on outputs associated to preconditions via the *realizedBy* construct. All unavailable outputs are added as goals for the next iteration. A recursive call is made to the compose algorithm with the new set of goals. Composition is successful if all goals are satisfied. Composition fails if the system is unable to find services that achieve some goal. Use of planning for composition has been discussed in detail in [WSH+03, SJF03, She04].

It is possible that the composer finds multiple operators/services that achieve the same goals/sub-goals. These are taken into account as contingencies. If the first choice fails to produce the intended result, the alternative services are tried. The first choice is determined using the user's preference (for example, the user prefers services that accept debit cards) or her context (for example, the user would prefer the fastest pizza delivery service when she is at work).

To further clarify our ideas, we illustrate the use of composition using a scenarios described below.

11.4.2 Scenario

The scenario involves finding a pizza delivery service and ordering a pizza. To explain the discovery process let us assume that the semantic directory uses an ontology based on the NAICS. We query the main directory using *Pizza_Delivery_Shops*. *Pizza_Delivery_Shops* is a *subClassOf Limited_Service_Restaurants* in the NAICS based ontology. The directory returns a large number of ranked matches. Services classified as *Pizza_Delivery_Shops* get a higher ranking compared to services classified as *Limited_Service_Restaurants*. Options are further narrowed down using the user's context (in this case the location) ascertained from the eWallet. Assume that agent finds a single pizza delivery shop near its current location.

The main service description returned by the directory look-up, points to the *Process* that contains the atomic processes. We synthesize planning operators from these descriptions and try to build a plan that will enable the user to order a pizza and get it delivered to his current location. Figure 44 shows the operators for the pizza delivery service.

The LocationCheck service ensures that the pizza service delivers to the use's location. CreditCardInfo service validates the credit card. DeliveryAdd service obtains the user's street address. Finally, the PizzaBuy service lets the user buy the pizza.

The user request for getting the pizza delivered is transformed into a goal- PizzaBought. The PizzaBuy service achieves this goal and is therefore included in the plan. The PizzaBuy service has two preconditions and an input. The preconditions are associated to the corresponding ouputs using the *realizedBy* predicate. These outputs along with the single input- PizzaType are made the goals for the next iteration. Other operators are included in the plan in a similar fashion. In the end, all goals are checked against the user's eWallet. Missing information like PizzaType is captured by prompting the user.

```
;Pizza Delivery Service
; s: service name
; i: input
; o: ouput
; p: precondition

(s LocationCheck
(i pin)
(o RangeCheck))

(s CreditCardInfo
(i CreditCardNumber)
(i CreditCardType)
(i CreditCardExpDate)
(o CreditCardStatus))

(s PizzaBuy
(p CCAuth)
(p DeliveryAddReq)
(i PizzaType)
(o PizzaBought))

(s DeliveryAdd
(p InRange)
```

Figure 44. Pizza delivery operators.

Figure 45. shows the plan generated by the planner. Once all the information is available, the plan can be invoked by accessing the WSDL descriptions via the OWL-S grounding.

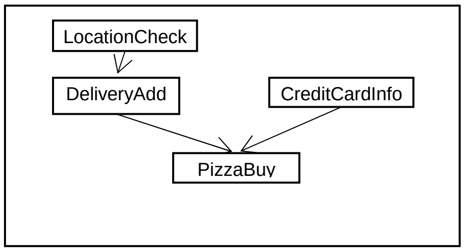


Figure 45. Plan for ordering a pizza.

11.5 Concluding Remarks

In this paper, we provided an overview of *my*Campus, a semantic web environment aimed at enhancing everyday campus life. Within *my*Campus, users over time acquire or subscribe to a variety of task-specific agents that assist them in the context of different tasks (e.g. scheduling meetings, sharing documents, organizing evenings out, filtering and routing incoming messages, etc.). Many of these agents require knowledge of the context within which their user operates as well as possibly information about the context of other users. In *my*Campus, sources of contextual information (e.g. calendar, location tracking functionality, organizational information, etc.) are represented as Semantic Web Services. These are described by profiles that can refer to any number of relevant ontologies. Service descriptions also include information about how to invoke a service. The end result is an environment, where relevant sources of contextual information about a user can automatically be discovered and accessed in support of different queries. This approach makes it possible to accommodate users that rely on different sets of contextual resources (e.g. different calendar systems, different sources of location information, etc.), and to adapt to situations where sources of contextual information for a user may change over time (e.g. different location tracking services depending on where the user is). Queries about a given user's context are submitted to that user's e-Wallet, which acts as a gatekeeper and a clearinghouse for the user's personal information, enforcing user-specified privacy rules. As users subscribe to or acquire new task-specific agents, these agents can find relevant contextual information about users by querying their e-Wallets.

*my*Campus agents can range from simple agents that rely on one or more sources of contextual information about their users to more complex agents that are capable of dynamically building plans in response to requests from their users. In this paper, we detailed how Semantic Web Services in *my*Campus can be used to support: (1) The dynamic discovery and access of contextual sources of information about a user via her e-Wallet (2) The automated generation of plans by task-specific agents through the discovery of services modeled as planning operators that can be dynamically composed to satisfy one or more user-goals.

12 Appendix D: Overview of a Museum Tour Guide Application

This appendix is based on our article on “Semantic Web Technologies for Context-Aware Museum Tour Guide Applications” [CHGS05].

Abstract

Traditionally, visitors to museums have been left having to choose between finding their way around exhibits on their own or taking a standardized group tour with a guide. In this paper, we describe a context-aware museum tour guide that adjusts its recommendations to the interests and contexts of individual visitors and enables them to selectively share their experience with others. The tour guide is built around an innovative Semantic Web framework that minimizes the development and maintenance costs associated with the introduction of new exhibits, new visitor-oriented services and new sources of contextual information. In particular, it features a semantic web rule reasoning engine that enables visitor-oriented services to identify relevant sources of contextual information and to enforce user-specified privacy preferences about what information they are willing to share with others (e.g. “Only members of my group can see my current location”, or “Only my friends can see how I rate exhibits”). While still in prototype stage, the tour guide’s target environment is the National Museum of Natural Science, one of Taiwan’s largest museums with over 3 million visitors per year.

12.1 Introduction

The traditional museum visitor experience has been characterized by having to choose between a limited number of predefined guided tours and the challenge of visiting on one’s own. In this paper, we detail work on a knowledge-rich, context-aware museum tour environment aimed at ***providing visitors with a more engaging and more personalized experience*** [Coop93]. The environment revolves around a Semantic Web framework built to capture and exploit knowledge about the museum’s exhibits, its layout as well as contextual information about visitors (e.g. their interests, their locations, how much time they have available, whether they are visiting as part of a group, etc.) and relevant privacy preferences. We discuss key features of our architecture, focusing on how they improve over prior efforts to develop context-aware museum tour guides and how they also extend earlier Semantic Web research. In particular, our use of Semantic Web technologies facilitates the development and refinement of new domain ontologies, whether to describe new exhibits, visitor interests or new sources of contextual information. Our reasoning technology allows for the definition of rules that refer to these ontologies, whether to specify ***application-specific preferences*** (e.g. “If I have some extra-time, please tell me about places where I can eat at the museum”) or ***privacy preferences*** (e.g. “Only show my location to my teacher”).

As they enter the museum, visitors are given a PDA that they carry with them for the duration of their tour. As a first step, the PDA asks them a few simple questions to understand their interests and relevant preferences, including some privacy preferences. This process has been kept as light as possible – to minimize burden on the user. It only takes a couple of minutes. Based on this information, which includes information about the user's interests, age, and how much time he or she has available, the guide builds an initial tour. This tour is used to give directions to the visitor and is adjusted, as needed, as the tour progresses. For instance, if the visitor falls behind, the tour guide will tell him so and may recommend dropping one or more exhibits from the current tour. On the other hand, if the visitor has extra time, the guide may suggest additional exhibits or possibly offer the user to take a break for a snack at a nearby restaurant.

Our museum tour guide is not built as a single monolithic application. Instead, it is an environment aimed to support a growing collection of applications that share access to information about the visitor's interests and context subject to individual privacy preferences. This includes both pull and push applications. An example of a push application can come in the form of suggestions sent to a visitor recommending that he or she takes a look at a nearby exhibit. An example of an equivalent pull application is one where the visitor can ask for a similar recommendation. The environment also supports peer-to-peer interactions between users, enabling them to find one another and share comments and ratings about different exhibits.

Within our architecture, each user has an eWallet that serves as a clearinghouse and gatekeeper to resources available for that user. This includes both information resources about his interests, contextual attributes and exhibition ratings/comments as well as communication resources such as a display for push messages. Applications that seek to access any of this information submit queries to the visitor's eWallet. The eWallet checks the user's privacy preferences to determine whether the query is compatible with the user's preferences and, if it is, returns an answer or activates a service (e.g. displaying a push message). Privacy preferences currently supported by the eWallet include ***access control preferences*** (e.g. "Only members of my class can see my location"), ***obfuscation preferences*** (e.g. "Members of my group can see the exhibit I am at but other visitors can only see the building I am in"). They can also include ***disruption preferences*** (e.g. "Don't send me promotional messages about items for sale at the museum store").

Museum tour guide applications as well as visitor eWallets are implemented on top of a ***Semantic Web Rule Engine (SWRE)*** that generalizes some of our earlier work in MyCampus, a semantic web environment aimed at enhancing everyday campus life with context-aware applications [GS04a]. This architecture would make it possible to eventually extend the current museum tour guide environment to other tourism, education or sightseeing applications, enabling these different applications to leverage a common set of contextual resources and user preferences.

The remainder of this article is organized as follows. Section 2 provides a brief literature review. Section 3 outlines the architecture of our Semantic Web environment for context-aware museum tour guide applications. Section 4 briefly outlines relevant ontologies and rules and describes our Semantic Web Rule Engine. Section 5 outlines scenarios aimed at illustrating the operation of our environment. Section 6 provides a summary along with some concluding remarks.

12.2 Brief Literature Review

Prior efforts to develop context aware applications are many. Early work in context awareness includes the Active Badge System developed at Olivetti Research Lab to redirect phone calls based on people's locations [WHFG92]. The ParcTab system developed at the Xerox Palo Alto Research Center in the early nineties relied on PDAs to support a variety of context-aware office applications (e.g. locating nearby resources such as printers, posting electronic notes in a room, *etc*.) [Schil95, Schil94]. Other relevant applications that have emerged over the years range from location-aware tour guides (e.g. HIPS [OS00]) to context-aware memory aids. More recent research efforts in context awareness include MIT's Oxygen [Der99], CMU's Aura [GSSS02] and several projects at Berkeley's GUIR (e.g. [HL01]) to name just a few.
While early context-aware applications relied on *ad hoc* architectures and representations, it was quickly recognized that separating the process of acquiring contextual information from actual context-aware applications was key to facilitating application development and maintenance. Georgia Tech's Context Toolkit represents the most significant effort in this direction [DA99, DSFA00]. In the Context Toolkit, widgets act as wrappers that provide access to different sets of contextual information (e.g. user location, identity, time and activity), while insulating applications from context acquisition concerns. Each user (as well as other relevant entities such as physical objects or locations) has a context server that contains all the widgets relevant to it. This is similar to our notion of e-Wallet, which serves as a directory of all personal resources relevant to a given user (e.g. relevant location tracking functionality, relevant collections of preferences, access to one or more calendar systems, *etc*.). Our Semantic e-Wallet however goes one step beyond Dey's Context Toolkit. It makes it possible to leverage much richer models of personal resources - what personal information they give access to, when to access one rather than the other, how to go about accessing these resources. In addition, it includes access control and obfuscation functionality to enforce user privacy preferences.

This richer model was first demonstrated in MyCampus, a Semantic Web environment for enhancing everyday campus life through an open collection of context-aware applications [SCV+03, GS04a]. Our notion of Semantic eWallet also extends current eWallet technologies by providing a unified privacy front-end to all the resources available for a given user. It also extends work on languages such as P3P/APPEL [P3P], EPAL [EPAL], SAML) [SAML], or XACML [XACML] by allowing for the definition of policies that refer to an open collection of ontologies and by allowing for context-sensitive policies, including obfuscation policies.

12.3 System Architecture

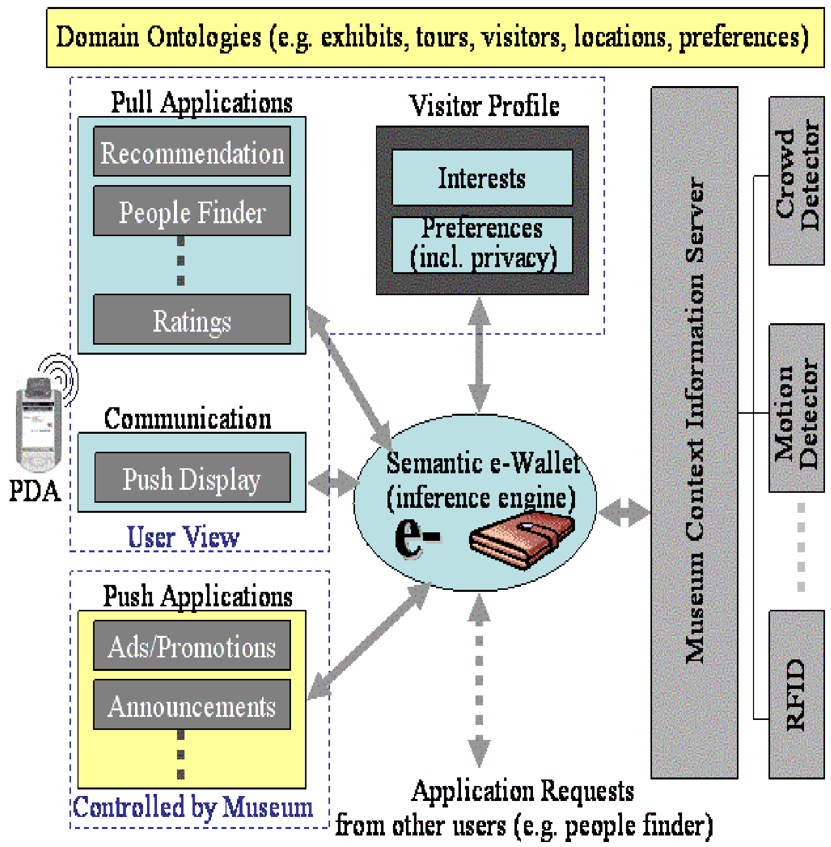

Domain Ontologies (e.g. exhibits, tours, visitors, locations, preferences)
Pull Applications
Recommendation
People Finder
Ratings
Visitor Profile
Interests
Preferences (incl. privacy)
PDA
Communication
Push Display
User View
Semantic e-Wallet (inference engine)
e-
Push Applications
Ads/Promotions
Announcements
Controlled by Museum
Application Requests from other users (e.g. people finder)
Museum Context Information Server
Crowd Detector
Motion Detector
RFID

Figure 46. Museum Guide Architecture

Each user interacts with an open collection of applications via a PDA he receives upon entering the museum. As a first step, the user is asked a few questions aimed at identifying his interests as well as relevant preferences, including privacy preferences. This information is stored in a visitor profile (Figure 46). Applications available to the user include both pull and push applications. An example of a pull application is a recommendation service to suggest additional exhibits to the user, based on his interests, exhibits he has already seen, his current location, and how much time he still has available. Other examples include a service to provide users with directions on how to go to a particular exhibit, a people finder to locate other visitors, a rating application that enables visitors to share ratings and comments about exhibits. Examples of push applications include public announcement messages (e.g. shop closing times, movie start times, museum closing times, etc.) and possibly ads/promotions (e.g. telling a visitor that a book about an exhibit he is visiting is available at the museum store). Requests to send push messages to the user or requests to access information about the user (e.g. his preferences or one or more of his contextual attributes such as his location) are directed to a Semantic eWallet. The Semantic eWallet, which is described in more detail in Section 4 and 5, is a ***reasoning engine*** that enforces the user's privacy preferences. This includes enforcing preferences about who/what application has the right to access what information about the user (and at what level of granularity) and what types of push messages the visitor is willing to accept. The visitor's interests and his preferences, including privacy preferences, are expressed in relation to concepts defined in OWL ***domain ontologies*** [OWL]. The visitor's profile, his push display application as well as the infrastructure used to monitor the user's context are represented as web services that can be invoked by the Semantic eWallet. As such the Semantic eWallet serves as a ***unified front-end to resources available for the visitor***, acting both as a ***clearinghouse*** and ***a gatekeeper to these resources***.

Details about the infrastructure used to monitor and access contextual information are provided in [MJH+04]. It revolves around an SQL query interface to an open collection of sensor services. Current sensor services include RFID tags and readers used for people location tracking. Motion detectors are used to monitor visitor streams. The infrastructure also supports higher level services such as services to compute the distance from a visitor's current location to a particular exhibit.

12.4 Semantic Web Technologies for Privacy and Service Invocation

Applications, user profiles and contextual information in our museum tour guide environment refer to an open collection of ontologies expressed in OWL [OWL]. These ontologies define relevant domain concepts such as exhibits, tours, exhibit topics, contextual attributes, etc. For instance, Figure 47 provides a simple definition of the concept of museum exhibit in OWL and Figure 48 shows an instance of this concept (in RDF).

We have also developed ROWL, an extension of OWL, to specify rules that relate to concepts defined in our ontologies [GS04b]. In ROWL, a rule is expressed as consisting of a *body* (i.e. a set of conditions) and a *head* (i.e. an action to be taken if the conditions are satisfied). Rules in ROWL can include variables that refer to instances of concepts defined in ontologies. A simple example of a rule is illustrated in Figure 49, indicating the preference of a particular visitor, Ben, to have an extra stop inserted in his tour, if his "early_tardy_status" is "early", namely if he has some extra time. A variation of this rule for another visitor could indicate that he would rather stop at a restaurant and eat, if he has some time left.

```
<owl:Class rdf:ID="Exhibit"/>
<owl:DatatypeProperty rdf:ID="name">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="topic">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://iii.org.tw/ontology/Topic#Topic"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="recommended_duration_time">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="CrowdSize">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="TargetAgeGroup">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="interactivityLevel">
	<rdfs:domain rdf:resource="#Exhibit"/>
	<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</owl:DatatypeProperty>
```

Figure 47. Simplified concept of museum exhibit.

```
<exhibit:Exhibit rdf:ID="exhibit_Ofd10231">
	<exhibit:name>Our Galaxy</exhibit:name>
	<exhibit:topic rdf:resource="&topicins;#astronomy 124"/>
	<exhibit:recommended_duration_time>10</exhibit:recommended_duration_time>
	<exhibit: CrowdSize >nil</exhibit: CrowdSize>
	<exhibit:TargetAgeGroup>elementary school</exhibit:TargetAgeGroup>
	<exhibit: interactivityLevel >for watch</exhibit: interactivityLevel >
</exhibit:Exhibit>
```

Figure 48. An instance of a museum exhibit.

Our rule ontology includes more specialized rules such as privacy rules and service invocation rules. An example of a privacy rule is illustrated in Figure 5. Privacy rules are defined using three tags [GS04a]: a *content* tag describes the piece of knowledge to which the privacy rule applies (e.g. a visitor’s location); a *check* tag specifies conditions that need to be met to authorize read access (e.g. to allow someone to see the visitor’s location), and an optional *revision* tag is used to specify possible obfuscation rules, e.g. to specify that location information can only be disclosed at the level of the building the user is in but not at the level of the particular exhibit he is at. Figure 50 illustrates one such rule where someone only allows his teammates to see his location at the level of the exhibit (“stop”) he is at.

```
<rowl:Rule rdf:ID="rule1">
        <rdfs:label>Insert a new exhibit</rdfs:label>
        <rowl:body rdf:parseType="Collection">
            <visitor:Visitor rdf:about="#VisInst">
                <visitor:name>Ben</visitor:name>
                <visitor:tour rdf:resource="#tour"/>
                <visitor:interest rdf:resource="#pref"/>
                <visitor:extra_time_preference>visit</visitor:extra_time_preference>
                <visitor:early_tardy_status>early</visitor:early_tardy_status>
            </visitor:Visitor>
            <exhibit:Exhibit rdf:about="#e">
                <exhibit:topic rdf:resource="#pref"/>
            </exhibit:Exhibit>
            <tour:Tour rdf:about="#tour">
                <tour:stop>
                        <stop:Stop rdf:about="#stop">
                                <stop:nextstop>nil</stop:nextstop>
                        </stop:Stop>
                </tour:stop>
            </tour:Tour>
            <stop:Stop rdf:about="#stop1">
                <stop:stop_at rdf:resource="#e"/>
            </stop:Stop>
        </rowl:body>
        <rowl:head rdf:parseType="Collection">
            <tour:Tour rdf:about="#tour">
                <tour:stop>
                        <stop:Stop rdf:about="#stop">
                                <stop:nextstop rdf:resource="#stop1"/>
                        </stop:Stop>
                </tour:stop>
                <tour:stop>
                        <stop:Stop rdf:about="#stop1">
                                <stop:stop_at rdf:resource="#e"/>
                                <stop:previousstop rdf:resource="#stop"/>
                        </stop:Stop>
                </tour:stop>
            </tour:Tour>
        </rowl:head>
</rowl:Rule>
```

Figure 49. Expressing visitor preferences as ROWL rules.

```
<sowl:PrivacyRule>
  <rdfs:label>people can only know the Location of teammate</rdfs:label>
  <sowl:target>
    <visitor:Visitor rdf:about="&variable;#VisInst">
      <stop:Stop_at rdf:resource="&variable;#stop_at"/>
    </visitor:Visitor>
  </sowl:target>

  <sowl:check>
    <rowl:And>
      <rowl:condition>
          <qowl:Query rdf:about="&variable;#query">
            <qowl:sender rdf:resource="&variable;#teammate" />
          </qowl:Query>
      </rowl:condition>
      <rowl:condition>
          <visitor:Visitor rdf:about="&variable;#VisInst">
              <visitor:Teammate>
                  <visitor:visitor rdf:resource="&variable;#teammate"/>
              </visitor:Teammate>
          </visitor:Visitor>
      </rowl:condition>
    </rowl:And>
  </sowl:check>

  <sowl:revision>
    <visitor:Visitor rdf:about="&variable;#VisInst">
        <stop:Stop_at rdf:resource="&variable;#stop_at"/>
    </visitor:Visitor>
  </sowl:revision>
</sowl:PrivacyRule>
```

Figure 50. Example of a Privacy Rule.

Within our environment, queries supported by the Museum Context Information Server as well as resources such as displaying push messages on the visitor's PDA are modeled as web services that can dynamically be invoked by rules. For instance, a request to display a promotion message on the visitor's PDA is directed to that visitor's eWallet. Upon receiving such a request, the eWallet checks the request against the visitor's privacy preferences (in this case disruption preferences). If the request is compatible with the visitor's preferences, it gets displayed on his PDA by invoking the push display application as a service. Similarly, rules that require information about the user's context such as his location result in the invocation of services supported by the Museum Context Information Server.

Technology to reason about facts and rules has been implemented in two different forms:

- A generic ***Semantic Web Rule Engine (SWRE)*** that can be used as a basis for developing tour guide applications (e.g. an application to recommend additional stops to the visitor using rules such as the one illustrated in Figure 49)

- A ***Semantic eWallet***, which provides a unified front-end to resources available for a given visitor subject to his privacy preferences. This includes access to the visitor's contextual information and preferences as well as access to his PDA to display push messages. The Semantic eWallet can be viewed as a specialized version of the SWRE that supports both privacy and service invocation rules.

Both reasoning engines have been implemented in JESS [JESS], using XSLT transformations to translate RDF/RDF-S, OWL and ROWL annotations into CLIPS [GS04a]. The engines also support rollback points that make it possible to retract and modify facts (e.g. removing a stop from a visitor's tour). By relying on an open collection of ontologies, by modeling preferences as rules expressed in terms of these ontologies and by modeling contextual resources and applications as web services, our museum tour guide environment makes it easy to introduce new exhibits, new preferences, new sources of contextual information and new applications.

12.5 Illustrative Scenarios

Visitors are insulated from the complexity of the underlying Semantic Web infrastructure by simple menu-based interfaces. Upon entering the museum, they are prompted to specify their interests by selecting from a list of topics. They are also asked to indicate how much time they have available and to specify a few simple preferences, such as whether they plan to eat at the museum. Privacy preferences are currently specified using three slider bars: one for museum-operated push services, one for visitors that are part of the same group, and one for other visitors. Each slider bar requires visitors to select from three predefined privacy profiles: low, medium or high. Once selected, these options are translated into rules and facts that refer to the museum's ontologies and are stored in the visitor's profile (Figure 1). Based on this profile, an initial tour is developed for the visitor. The tour is used to provide directions to the visitor and is updated during the course of his visit based on contextual information such as whether the user has fallen behind or based on crowds at different exhibits. The following illustrates two scenarios supported in the current prototype implementation.

12.5.1 People Finder Application

Consider two visitors, Ben and Ted, who are part of a same group. They have different interests. Accordingly, their individual museum tour guides have developed different tours for them. Ted has specified that he is willing to let members of his group see his location. This preference has been translated into a rule similar to the one shown in Figure 50 and is stored in his individual profile. As lunch time is drawing near, Ben decides to look up his friend, Ted, using the People Finder application available as a button on the screen of his PDA. Ben's People Finder application sends a request to Ted's e-Wallet, which in turn checks it against Ted's privacy preferences. The eWallet determines that Ben is a member of Ted's group and that therefore he is allowed to see his location. The eWallet proceeds and invokes the Museum Context Information Server, requesting Ted's location. Upon receiving the location back, the eWallet forwards it to

Ben’s PDA. Ben asks his PDA to give him directions to Ted’s current location so that the two of them can go and have lunch together.

12.5.2 Dynamic Recommendation Application

Ted and Ben end up having a rather quick lunch together. As he resumes his tour, Ted is told that he has some extra time and is asked whether he would like a recommendation for another exhibit he might want to look at. Ted accepts, which triggers his recommendation application to look for a nearby exhibit that matches his interests and does not take longer than the extra time he has available. This recommendation involves accessing a special service supported by the Museum Context Information Server to compute the time it takes to walk between different places.

12.6 Summary and Concluding Remarks

In this paper, we introduced a context-aware museum tour guide that adjusts its recommendations to the interests and contexts of individual visitors and enables them to selectively share their experience with others. The tour guide is built around an innovative Semantic Web framework that minimizes the development and maintenance costs associated with the introduction of new exhibits, new visitor-oriented services, new sources of contextual information and new preferences. In particular, it features a semantic web rule reasoning engine that enables visitor-oriented services to invoke relevant sources of contextual information and to enforce user-specified privacy preferences (e.g. “Only members of my group can see my current location”, or “Only my friends can see how I rate exhibits” or “I don’t care for promotional messages”). While still in prototype stage, the tour guide’s target museum is the National Museum of Natural Science in Taiwan, a museum that welcomes over 3 million visitors per year.